TURING 图灵程序设计丛书

全套设计方法和流程，拿来就用，完全干货，绝对实战
配合EnhanceJS框架，彻底实现面向未来的Web设计

渐进增强的Web设计

[美] Todd Parker [英] Patty Toland
[英] Scott Jehl [法] Maggie Costello Wachs 著
牛化成 译

Designing with Progressive Enhancement

Building the Web that Works for Everyone



人民邮电出版社
POSTS & TELECOM PRESS

数字版权声明

Todd Parker

Filament集团公司创始人，具有12年以上创建高度实用、可访问和直观界面的经验。Todd曾在Zefer公司和CSC顾问公司担任首席体验设计师。*Communication Arts*杂志对Todd卓越的设计工作给予了充分肯定。

Patty Toland

Filament集团联合创始人，具有20多年的企业和机构客户咨询经验。Patty的专长是制定稳健的信息和传播策略，包括品牌推广、成分分析、信息架构和系统设计，以及文字创作。Patty曾供职于Zefer公司、Kohn Cruikshank公司和哈佛商学院。

Scott Jehl

设计和开发技艺俱佳。加入Filament集团之前，Scott曾为《新英格兰医学期刊》、Footjoy Golf和阿斯彭/斯诺马斯度假村等客户提供过服务。他还运营着在线网站地图工具WriteMaps.com。此外，他还是jQuery设计团队成员。

Maggie Costello Wachs

Filament集团的编程和产品负责人。Maggie在前端编程领域有着高超的专业水准，特别致力于编写兼容标准的标记、CSS和脚本，以实现与渐进增强的最佳搭配。加盟Filament集团之前，Maggie曾供职于Monitor集团和Zefer公司。



图灵程序设计丛书

渐进增强的Web设计

[美] Todd Parker [英] Patty Toland
[英] Scott Jehl [法] Maggie Costello Wachs 著
牛化成 译

Designing with Progressive Enhancement

Building the Web that Works for Everyone

人民邮电出版社
北 京

图书在版编目（CIP）数据

渐进增强的Web设计 / （美）帕克（Parker,T.）等著；
牛化成译. -- 北京 : 人民邮电出版社, 2014.1
（图灵程序设计丛书）
书名原文: Designing with progressive enhancement:building the web that works for everyone
ISBN 978-7-115-33839-6

Ⅰ. ①渐… Ⅱ. ①帕… ②牛… Ⅲ. ①网页制作工具—程序设计 Ⅳ. ①TP393.092

中国版本图书馆CIP数据核字(2013)第284873号

内 容 提 要

随着互联网技术的不断发展壮大，网络覆盖率和可联网设备不断增加，用户对可访问性的需求也与日俱增。本书是一本网页设计与开发方面的实用指南，介绍了一种渐进增强的编程方法，利用 JavaScript、高级 CSS 和 Ajax 制作能实现高度交互体验的网站，同时还确保代码库无需修改就能到处运行。这个法则简单地说就是建议所有的网站内容和功能都以语义化的 HTML 为基础，让任何具备网络功能的设备都可以使用，然后再在上面无缝叠加基于高级 CSS 和 JavaScript 的增强功能。

本书适合网站设计和开发人员阅读。

◆ 著　　[美] Todd Parker [英] Patty Toland [英] Scott Jehl
[法] Maggie Costello Wachs
译　　牛化成
责任编辑　刘美英
执行编辑　李　静
责任印制　焦志炜

◆ 人民邮电出版社出版发行　　北京市丰台区成寿寺路11号
邮编　100164　　电子邮件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
北京　　　　　印刷

◆ 开本：800×1000　1/16
印张：20.5
字数：485千字　　　　2014年1月第1版
印数：1－3 000册　　　2014年1月北京第1次印刷

著作权合同登记号　图字：01-2011-3257号

定价：69.00元

读者服务热线：(010)51095186转600　印装质量热线：(010)81055316

反盗版热线：(010)81055315

广告经营许可证：京崇工商广字第 0021 号

版 权 声 明

致　　谢

首先，我们想感谢 Peachpit/New Riders 给我们这个机会并鼓励我们写这本书。多年来，我们一直在自己的项目和网站开发中考虑可访问性，并且使用测试驱动的渐进增强方法。但是，直到 Peachpit 找到我们，我们才有意识地将有关这一重要主题的原则和技巧综合在一起，清晰地表达出来。仅就给我们带来的益处而言，运用这些原则和技巧的结果已经很不可思议了。我们对在此过程中获得的那些无价反馈深表感谢。

特别感谢 Peachpit/New Riders 的两位 Wendy：Wendy Sharp 找到了我们的在线实验室，联系并邀请我们写书，还就如何通过这一媒介分享我们的想法提供了宝贵的意见；而正因为有了 Wendy Katz 对本书仔细且周到的编辑和无尽的幽默、机智和耐心，我们才能一路走到终点。另外还要感谢 Glenn Bisignani 的市场营销见解和支持，以及 Aren Howell、Mimi Heft 和 David Van Ness 的精美设计。

深深感谢技术编辑 James Craig。他在可访问性和 ARIA 上的专业建议大大提升了本书相关内容的质量。诸多有关可访问性的好建议多半有他的功劳。（特别指出，如果我们哪点做得不好，责任全在我们自己。）

感谢 John Resig 和 jQuery 及 jQuery UI 团队每个成员的支持，以及这个让我们爱不释手、不可思议的 JavaScript 库（不仅这本书一直在用它，我们的日常工作中也是如此）。还要特别感谢 Brandon Aaron 参与并显著改进了我们 EnhanceJS 框架的性能。

我们想要感谢尊敬的客户和同事，是他们激励我们在工作中思考这些重要的想法，并提出他们在现实世界中的问题，从而激发出本书中的许多解决方案。感谢 Steve Krug 的建议和鼓励，以及 Derek Jones 在本书印制过程中提供的帮助。

我们还想对许多组织表示谢意，它们无比慷慨地提供了一个公开的论坛，让我们分享、讨论并完善了本书收录的许多想法和技巧：*A List Apart Magazine*、Delve UI 大会、MIT WebPub、Markup & Style Society、jQuery、Ajaxian 和 *Smashing Magazine*。

本书离不开无数慷慨睿智之人公开分享和私下传授的想法、技巧和代码，这些人包括：Tony Aslett、Doug Bowman、Andy Budd、Corey Byrnes、Dan Cederholm、Tantek Çelik、Steve Champeon、Andy Clarke、Jeff Croft、Derek Featherstone、Seth Ferreira、Nick Finck、Becky Gibson、Brandon Goldsworthy、Scott González、Marc Grabanski、Aaron Gustafson、Chris Heilmann、Dorothy Hesson、Molly Holzschlag、Shaun Inman、Paul Irish、Yehuda Katz、Jeremy Keith、Peter-Paul Koch、Bruce Lawson、Ethan Marcotte、Michael McGrady、Eric Meyer、Cameron Moll、Jon Reil、Richard Rutter、

Travis Schmeisser、Dave Shea、Chris Smith、Jonathan Snook、Krista Stevens、Nicole Sullivan、Karl Swedberg、Randy Taylor、Dan Webb、Richard Worth、Jön Zaefferer、Jeffrey Zeldman，还有无数其他人。

许多才能卓著、思想深刻的人阅读我们实验室的博客并加以评论，他们帮助构建和完善了本书的许多技巧。十分感谢你们的鼓励、专业技术意见和批评。更重要的是，我们希望你们一直秉持这种理念，参与评论，贡献代码，这些代码可构成书中的演示示例以及可下载的插件。（欢迎前往 www.filamentgroup.com/dwpe 参与讨论。）

个人致谢

我们一致感谢 Christine Fichera 慷慨地将她的照片供本书使用。Todd 还特别感谢 Christine 和 Nathan 容许占用为他们洗澡和讲故事的时间。

Patty 感谢出色的导师及楷模 Judy Kohn 和 Jeff Cruikshank，感谢 Toland 一家无尽的耐心和支持，哪怕他们至今仍然没有完全理解她靠什么为生。

Scott 感谢他的妻子 Stephanie 和所有家人朋友在他写书时的耐心和鼓励，并感谢从第一天起就完全支持和鼓励他的父母。

Maggie 感谢丈夫 Michael、家人和朋友的大力支持，感谢他们的鼓励和好脾气，特别感谢母亲 Barbara Thompson 给了她一生的启迪，还要感谢众多为书中想法提供反馈的人。

引言：谈一谈渐进增强设计

从很多方面看，现在都是成为网页设计师的最佳时机。

新的网站和应用程序正在以惊人的速度涌现，它们改变了我们工作、交流和生活的方式。我们在网上经营生意，建立人际关系，表达意见，自学各种知识，还找到了各种各样的娱乐方式。网页标准的进步不断让网络变得更简单、更快捷、更动态和更强大。“在线”这个词已经摆脱了“线”的束缚：具备上网功能的手机和小型上网本大量涌现，几乎让我们随时随地都可在线。

但是，这座巨大的互联网宝库存在一个问题：虽然高级交互功能一般可以在支持高级 CSS 和 JavaScript 功能的最新浏览器上完美运行，但是市面上无数种上网设备（从最新的 Kindle 或 Wii 游戏系统到各种老式计算机和手机）对这些功能只提供有限支持或者完全不支持，这可能会导致这些高级交互功能缺损或不可用。即使是最新的浏览器，如果网站开发者没有考虑对屏幕阅读软件和其他辅助设备的支持，网站就无法服务于盲人或视障用户。

作为网页设计和开发人员，我们总是在平衡三个相互有些冲突的目标。我们想使用眼前所有激动人心的新技术，实现引人入胜的交互体验。但同时我们也努力确保网站对所有人都是可访问、可用的。另外，从开发的角度看，尽可能编写最整洁、最容易维护的代码也同样重要。我们一直在寻找能在实际项目中优雅地实现所有这三个目标的开发道路。

令人高兴的是，我们觉得已经找到了这样一条路。本书是一本网页设计与开发方面的实用指南，特别注重介绍如何利用 JavaScript、高级 CSS 和 Ajax 制作能实现高度交互体验的网站，同时还确保代码库无需修改就能到处运行。我们的方法就是基于一种叫作*渐进增强*的编程法则，这个法则简单地说就是建议所有的网站内容和功能都以语义化的 HTML[①]为基础，让任何具备网络功能的设备都可以使用，然后再在上面无缝叠加基于高级 CSS 和 JavaScript 的增强功能。

渐进增强地开发网站并不意味着花费更多的精力，但它确实需要你调整开发流程和思路。一些人可能会说这么做不值，因为目前的开发方式能照顾到的对象已经覆盖了“大多数”用户，浏览器对网页标准支持的不断改进也将弥合剩下的用户缺口。

然而，近期的互联网应用与设备发展趋势告诉我们，事实*恰恰相反*：一些最前沿可上网设备的浏览器对网页标准支持不佳，同时，那些运行着过时、简陋版本浏览器的老式电脑和手机的数量不减反增。

① 语义化的 HTML（semantic HTML）鼓励将 HTML 标签用于体现网页的实际内容，把页面美化和外观调整的任务交给 CSS。——译者注

无论如何，事实就是如此，人们在日常生活中使用各种各样的浏览器、平台和设备，他们期望自己喜爱的网站和应用能在所有这些地方无缝运行。面对这样日益复杂的设备环境时，构建只能在少数特定浏览器上正常运行的网站是不现实的，特别是不要忘了——渐进增强设计能让你的网站到处运行，并为所有人服务。

不过，我们知道单凭这些无法说服持怀疑态度的人。所以接下来看看全球网络受众和现有技术的真实现状，并探究一下常见编程方法可能导致的问题（轻则导致小小的不便，重则导致功能完全不可用）。

前所未有的网络世界

我们很容易忘记互联网发展壮大的速度有多快，以及它对我们生活的改变多么明显。仅在 10 年前，网络访问还局限在相当狭窄的用户范围中，而且用户通常都用着相当一致的硬件。今天，互联网成了真正的全球现象，用户越来越多样，用户所使用的上网设备也五花八门。

通过追踪并了解现在谁在上网、通过什么途径上网，以及未来的增长最有可能来自何处，我们就能更好地制定计划，迎接这个网络无处不在、人人都会上网的新世界。

所有人都（或不久即将）上网了

20 世纪 90 年代中期，互联网浪潮兴起之初，大多数互联网用户都居住在更为富裕的西方国家，他们受过教育（因为当时上网需要一定的技术能力），而且很可能是专业人士或者富人，因为装载系统的 PC 价格远远超出了普通人的承受能力，况且还需要昂贵的包月上网套餐和强大的基础设施。

在过去的 10 年中，计算机越来越好用，硬件、软件和上网费用的大幅下降也让网络触手可及。正因为如此，网络用户已经全球化，覆盖了所有的年龄段、教育程度和社会阶层，这使得网上的信息和工具更加大众化。

欣欣向荣的互联网普及进程不断吸引着世界各个角落的人上网冲浪。全球互联网用户的数量增长已经超过 450%。从 2000 年 12 月的 3.61 亿人增加到 2013 年 10 月（www.internetworldstats.com/stats.htm，截至 2012 年 6 月 30 日）的 24 亿人（超过世界人口总数的 34%）。全球各地都出现了迅猛的增长：非洲互联网用户从 450 万增加到 1.6 亿，亚洲用户从 1.14 亿到 10.7 亿，欧洲用户从 1.05 亿到 5.18 亿（www.internetworldstats.com/stats.htm，截至 2012 年 6 月 30 日）。

在美国，互联网用户在过去 10 年间越来越多样化，超过 92%的 18 岁至 29 岁人群和近半数的中老年人都已上网，网民的教育和收入水平更是千差万别。互联网正迅速变得无所不在：超过 73%的美国成年人称自己使用互联网和电子邮件，50%以上的农村地区已经联网（根据 Pew 研究中心的“互联网与美国生活”项目：www.pewinternet.org），接入互联网的公立小学教室已达 93%（根据美国教育部全国教育统计中心 2006 年发布的“美国公立学校和教室的互联网接入情况：1994—2005”。NCES 2007-020）。

不过，最有意思的可能还是以下这些事实：虽然亚洲有 7.98 亿网民，但互联网渗透率还不

到总人口的 28%，非洲的 6690 万则不到 16%，而美国和大多数西欧国家基本都在 70%~80%之间。随着互联网接入程度的提高，最大的潜在增长区域更有可能来自于美国之外。相对于西方，东方和欧洲的用户不但有着不同的语言和文化，而且他们的设备、上网方式、基础设施和使用期望都会很不一样。

不断提升的用户期望

伴随着全球用户和可上网设备数量的巨大增长，今天的互联网在提供些什么方面也发生了同样显著的变化。创新的网页内容和交互手段加在一起，让用户对网站行为和功能的期望发生了根本性的改变。下面列举了自 2000 年以来，用户需求发生的一些变化。

- Amazon、eBay和Netflix等在线运营的成功企业引入了层次更丰富的网页内容、内嵌媒体、动态交互方法和沉浸式体验，为用户体验设置了新的标准。
- 用户自创内容的涌现（博客、YouTube、Facebook、Digg、Twitter等）让网站的内容和结构向大众化转变，而且还提高了用户的期望值：能够根据自己的需要创建、引用和有创意地调用网络内容。
- 实时互联网让用户期待获得新鲜到秒的即时更新，包括他们在Facebook上的社交网络、Twitter上的关注者和Google Docs或Google Wave上的协作者。这种期望促使网站行为发生了很大的转变，从基础的静态页面切换转向基于Ajax的环境，从而几乎不再需要进行传统的页面刷新。
- 网络应用程序尝试在浏览器里提供与桌面程序同等质量的用户体验，包括复杂数据可视化、拖放手势和丰富的交互手段。这类程序的兴起符合将软件作为网络服务提供的总体趋势，以代替在计算机上安装的传统形式。
- 现在的用户会使用工作场所和家里的多种桌面浏览器上网，出门时则使用手机等设备。他们期望可以用任何设备随时访问自己的数据。

这些新变化让互联网成为强大的全球平台，能在浏览器里提供类似桌面应用程序的特性和功能，使所有用户都能获得实时通信的能力。

不断增长的用户可访问性需求

在线人口明显变得越来越多样化，特别是年长用户的比例增加，再加上前沿网站里涌现出大量的新交互模型，影响网站设计的考虑因素就又多了一层。

随着年龄的增长，人的视力、听力和运动能力会下降或受损，而这一切都对他们能否顺利进行网上冲浪有重要影响。白内障、失明、听力障碍、关节灵活性下降、精细动作控制能力丧失或手部颤动等病症，会让一个传统网站面临难于使用或更严重的问题。

这些年长用户可能需要更大的字号和对比度，特别是在长时间阅读时。网站应当支持调整文本设置以满足他们的需求，这点至关重要。许多失明或视力重度受损的用户在他们的桌面电脑上利用屏幕阅读软件等辅助技术阅读应用程序内容和网页。顺利使用屏幕阅读器上网的关键是：有

合理组织的语义内容和完整的键盘支持，这样导航就无需视力或鼠标了。而那些灵活度受损的用户用键盘导航通常也会比用鼠标更有控制感。

1998 年，美国联邦政府通过了 508 条款（Section 508），这部复健法案（Rehabilitation Act）的修正案要求联邦机构确保自己的电子和信息技术能够供残障人士使用，并确立了针对生理和心理残疾人士数字媒体可访问性的国家标准。虽然这些规则仅适用于政府机构和为联邦政府提供商品和服务的企业，但 508 条款为向所有用户提供访问服务提供了一套清晰且可施行的标准，并已经作为一套事实上的法律标准同时被许多私人组织采用。

万维网联盟（World Wide Web Consortium，W3C）一直非常积极地通过 Web 可访问性倡导组织（Web Accessibility Initiative，WAI）以及相关项目创建 Web 可访问性规范。W3C 的“Web 内容可访问性指南”（Web Content Accessibility Guidelines，WCAG 2.0）既提供了宏观的原则，也给出了包含具体指南和标准的检查清单，以确保网站可被访问。“可访问富互联网应用”（Accessible Rich Internet Application，WAI-ARIA）规范提供了一套可以被添加进标记语言（比如 HTML）的属性，以一种能被屏幕阅读器理解的方式描述滑动条（slider）和树形控件（tree control）等高级 UI 组件。

这些进展综合起来既包括了明确的要求，又提供了一套工具让今天的用户不论身体状况如何都能上网获得完整的信息。所有的开发者都应该重视将辅助功能加入网站并在屏幕阅读器上测试，因为缺乏可访问性在用户眼里意味着歧视，在法律层面可能也是如此。

举个塔吉特公司的例子。2005 年初，美国盲人联合会（National Federation for the Blind，NFB）来到了塔吉特公司，提醒他们网站上的一些功能许多残障用户无法使用。塔吉特公司声称他们的实体店足以满足那些用户的访问需求。出于对塔吉特公司回应的不满，NFB 在 2006 年 2 月对该公司提起诉讼，援引 1990 年的美国残疾人法案指控其违法。经过数年的法庭斗争，该案在 2009 年 8 月达成和解，条件是塔吉特公司支付 600 万美元的和解金，负担 NFB 将近 380 万美元的法律费用，并且同意大幅修改其网站以引入辅助功能，NFB 还将定期对网站进行可用性测试。

虽然这样的诉讼很罕见，但这样的案例足以说明在设计和开发中应重视可访问性。创建通用可访问网站的原则和规范正在不断完善。随着网络逐渐大众化，我们设计和编写代码的目标必须是通用可访问性：一个不分语言、文化、年龄、身体状况或技术平台，所有人都能用的互联网。

移动互联网的兴起

随着网络受众、用户期望和在线内容标准的不断进化，用户上网的地点和方式也在发生变化，特别是出现了具备上网功能的手机、视频游戏系统以及专门用于上网的设备。

截至 2008 年，全球 14 亿互联网用户中足有 75%通过移动设备上网，而非桌面端的浏览器独占了 29%的全球互联网流量（Tomi Ahonen，*Thought Piece: Mobile Telecoms Industry Size*，2009，www.tomiahonen.com）。手机的普及速度比人类发明史上其他任何技术都要快，而我们才刚刚感受到它对网络设计方式的冲击。

动互联网早期的手机屏幕小，处理器速度慢，而且只理解 XML 的一种简化版本，名为 WML

（Wireless Markup Language，无线标记语言）。很少有开发者或用户期望网站能在这些设备上无缝运行。

第一波具备上网功能的“智能手机”在 2000 ~ 2002 年左右登上舞台，包括诺基亚 9210 Communicator、Palm Treo、第一代 RIM 黑莓和微软 Windows CE 设备，它们将用户对实时访问数据和功能的期待提升到了一个新水平。这些设备能访问标准 HTML 网站，甚至还初步支持了 JavaScript 和 CSS。很快，数以百计（毫不夸张）能访问“真正”互联网的不同型号手机涌现出来，引发了移动互联网革命。

截至 2009 年，主动手机订阅数已经达到 46 亿（“Explaining 4.6 billion mobile phone subscriptions on the planet”，Communities Dominate Brands blog，11/6/09，http://t.cn/zRljrXB），这一数字使当前互联网用户总数相形见绌。虽然不是所有这些移动设备都有智能手机那样的网络浏览器，但公允地说，移动端是一个巨大且还在增长的市场，建立网站时不能忽视。

甚至连“移动设备”的定义也在持续进化和改变，囊括了可上网的触屏平板电脑、微型笔记本电脑和其他电子设备。每种设备都有自己独一无二的一套浏览器特性、插件支持、内置字体、屏幕尺寸和交互标准，从 RIM 黑莓的拇指滚轮和键盘交互，到亚马逊 Kindle 的微型摇杆控制器，再到苹果 iPhone 的多点触摸交互模型，等等。

一大批像任天堂 Wii，索尼 Playstation 3 和微软 Xbox 这样的视频游戏系统都内置有网页浏览器，这些系统自作主张地将网站设计“改编”成可以用游戏控制器在 3 米之外的电视屏幕上浏览。其他的消费电子设备（电子书阅读器、电视机、家用电话机，甚至是收音机闹钟和冰箱）也正在加入网页浏览器。我们的设计将会出现在呈爆炸性增长的多样化设备上。

设备和浏览器的“保质期”越来越长

对每一台新购买的电脑、手机或者电子设备而言，它很有可能会取代一台老旧设备，后者则被回收到技术“食物链”的下游。数百万的手机和电脑仍然在工作，它们或者在家庭内传递，在学校和社区中心公用，或者被捐赠给各种社会组织，其中许多最终会通过捐赠给 NGO（Non-Governmental Organization，非政府组织或民间组织）在遥远的国家落脚。在物资回收和重新利用非常普遍的发展中国家里，绝大多数设备都被精心保养，使用时间远超西方国家中通常的“可使用”寿命。每过一年，家庭、图书馆、学校和网吧里运行“过时”浏览器的机器数量都在持续增长。

除了当前这一大堆轮替中的老式设备之外，同样还有一批运行在非主流操作系统上的“边缘”浏览器。基于 Unix 的操作系统经常被用于廉价电脑，比如上网本、微型笔记本和“每个孩子一台笔记本”（One Laptop Per Child，OLPC）项目里的 XO 电脑。高端技术用户经常使用基于 Unix 的电脑运行多种“非主流”浏览器（例如 Konqueror）以及 Lynx 这样基于纯文本的浏览器，后者与主流浏览器渲染页面的区别非常大。每一种浏览器单独来看可能只占据了微不足道的份额，但它们合起来形成了全世界数以百万计的网络访问者。

今天，庞大而多样的上网人群，正在转变的用户体验预期，以及爆炸性增加且需要我们支持

的可上网设备，导致我们设计和开发的复杂程度几乎呈指数级增长。

通常的开发方式只针对有最新和最流行桌面浏览器的目标群体，却忽略了其他人。这种方式肯定会给数百万潜在的读者、顾客、求职者、婚恋交友者或政治支持者造成缺损和无法使用的体验。同时我们也能理解，桌面浏览器、手机、游戏系统和其他专用设备的数量和种类太多，对于任何一名开发者而言，哪怕只在其中一小部分设备上进行网站测试和调试，都很有难度，因为时间和精力总是有限的。

如果你还不相信用标准的方式开发最先进的网站存在问题，就请参考现有网站下面这些失败的真实案例。

Web 2.0 的地雷

Web 2.0 网站和应用程序在过去 10 年爆炸性增长，其中许多特别有趣和迷人的功能来自 JavaScript 对富交互性的支持、Ajax 的动态数据能力和高级 CSS 特性。

这些进步伴随着过去 10 年里数十个不同桌面浏览器版本的发布，这还不包括近几年层出不穷的可上网设备和手机。虽然许多当代浏览器拥护 Web 标准，它们渲染网页的方式却并不统一，加在一起就要求人们运用一系列复杂的编程、测试和时不时的修补手段，才能让网站的显示和运行方式保持一致。

许多网页设计和开发的趋势正在破坏互联网的基石——普遍可访问。

浏览器的狭窄视野

为了能掌控现实的开发和测试，所有企业和个人开发者（包括我们自己）都将编程和测试的焦点集中在一组特定的浏览器上。从发行合约的角度看，这是合理的：客户要求我们（书面）承诺我们的显示和工作方式与设计完全一致，而我们知道不是每种浏览器都能处理一个现代网站设计所需的样式和脚本。因此我们合约的一部分规定，根据一组事先确定的浏览器或运行环境进行代码测试，具体的浏览器和环境视不同的项目而定。

但是，那些没有使用受支持浏览器的用户怎么办？是否可以将他们挡在网站外面，只显示一条请他们升级浏览器的消息？许多人不升级他们的浏览器，可能因为他们对技术的理解还不足以完成升级，或者是出于安全原因不被允许在他们的机器上安装软件（尤其是在企业环境里），或者是因为某个他们离不开的网络应用程序只能在旧式浏览器上运行，所以不得不用它。

你很容易把那些使用不受支持浏览器的用户看得无足轻重，因为他们的比例微乎其微，或者你认为他们一般都不够懂行或技术不熟练，因此不属于目标用户。但差劲的浏览器性能并不只限于过时的浏览器或者卢德派[①]：亚马逊的 Kindle 通常被认为是一款“高精尖”设备，用户属于很挑剔的早期使用者，但它的浏览器是黑白显示和基于文本的，仅仅“实验性”地提供有限 CSS

① 卢德派（Luddites）是 19 世纪英国的一群技术熟练的纺织工人，他们抗议工业革命带来的机械化使他们失业。后来泛指那些反对技术进步和产业调整的人。——译者注

和 JavaScript 支持。

通过检测用户代理（user agent）信息将访问限制于一组特定的目标浏览器，从原理上看是不可靠的。举个例子，流行的黑莓智能手机有许多型号都提供了一项偏好设置，允许用户将他们的浏览器标识为黑莓、火狐或者 Internet Explorer，目的就是为了绕过开发者设置的这些障碍。

这就造成了在一些情况下开发者误以为他们阻挡了所有的"坏"浏览器，从而可能会在 JavaScript 和 CSS 的使用上更加大胆，最终导致那些修改偏好设置以获得访问权的用户面临非常糟糕的体验。

受支持浏览器列表这个概念还过度简化了许多别的假设：开发者通常假设受支持的浏览器会开启所有功能，比如图像、CSS、cookies、JavaScript 和各种插件（比如 Flash、Java 或 Silverlight），而且所有这些开启的功能都会如他们期望的那样工作。如果某一位用户使用受支持的浏览器，但因为安全或隐私原因决定禁用 cookies 或 JavaScript 怎么办？如果另一位用户为了加快网页载入速度，在移动设备上禁用了 CSS 或图像又该怎么办？当开发者估计所选择的受支持浏览器列表将能覆盖 95%的网络总人口时，他们忽略了在某些情况下虽然用户理论上用着受支持的浏览器，却已禁用了一种对设计至关重要的功能。

在现实世界里，代码失效的频率很可能远比开发者知道（或者愿意承认）的要高，因为绝大多数测试都是在相对安全和受控的环境中进行的：一组有限的现代浏览器，使用默认设置，以"典型"的屏幕分辨率显示，字体大小也是默认值。而现实世界的不可预测性和多样性要复杂得多。只考虑一小组"目标"浏览器营造了一种虚假的安全感。这种"狭窄的视野"使我们无法真正看到所有那些需要我们支持的浏览器和设备，或想方设法使我们的代码更加兼容和无懈可击。

"需要JavaScript"可能会排斥许多用户

现在，很多网站的关键功能依赖于 JavaScript，比如用 Ajax 生成页面内容或者提交和验证表单，这就意味着如果脚本功能不可用，整个页面或功能可能就会彻底失效。

如果你搜索一下启用 JavaScript 的用户百分比统计数据，得到的结果会是 85% ~ 98%。W3schools 的浏览器统计网站是被引用得最多的数据来源，它估计大约有 5%的互联网用户禁用了 JavaScript。（根据当前的互联网使用率，这就相当于大约 8300 万人！）与我们合作的一些客户使用的是自定义的企业浏览器设置，里面出于安全原因选择性修改或禁用了一些 JavaScript 功能。这些浏览器会正常地"登记"为 JavaScript 已启用，但不是所有的网站或页面都能正常工作。

许多流行的 JavaScript 和 Ajax 驱动的网站，包括 Adobe 的电子商店、旅行预定网站 Kayak 和项目管理工具 Basecamp 直接需要 JavaScript 来实现核心功能，如果脚本功能没有启用则会显示一条错误消息。

这些错误消息通常告知用户需要启用脚本功能并使用受支持的浏览器。但是如果某人正在机场，需要查询别的航班或者某个项目的状态，用的却是不支持 JavaScript 的旧型号黑莓或 Palm Treo

手机怎么办？下载一款新的浏览器不具备可行性。

数量惊人的大型电子商务网站将带来收入的关键功能制作成只有在脚本功能可用时才能正常工作。在 2009 年秋季重新上线的 Sears.com 网站上，一位顾客搜索某个商品或导航至任何商品列表页（比如家庭→电器→微波炉→桌上型）而没有 JavaScript 时，本应显示商品的结果区域会保持空白，只有一个 Ajax 的“载入中”动画。

这张页面只输出了一个用于占位的旋转动画，很明显是打算在页面载入后，通过一个 Ajax 请求获得一个商品列表来替换它，而刚载入时的页面上没有任何有意义的标记。搜索过滤器和推荐商品版块也依赖 JavaScript。对那些没有脚本功能的用户而言，Sears 的网站完全违背了它的主要目的：帮助人们搜索和购买商品。

假设这只是一种孤立情况，还能让人得到些许安慰，但是事实上这个问题广泛存在。在目前的沃尔玛网站（Walmart.com）上，每个“添加到购物车”按钮都是用 Ajax 添加到页面上的，没有 JavaScript 就不会有与购物相关的按钮。Toys R Us 和 The North Face 两家网站的商品详情页上都有一个“添加到购物车”按钮，但是点击那个按钮会调用一个 JavaScript 函数，如果浏览器禁用了脚本功能就不会执行任何操作。

所有这些创收机会都可以避免错失：只需在页面上放一个可用的“添加到购物车”按钮，然后提交一张简单的表单即可。

假定CSS支持是另一个潜在的故障点

假定浏览器支持 CSS（无论是否同时支持 JavaScript），同样会引入另外一大堆潜在问题。

许多旧版桌面浏览器渲染 CSS 时并不遵循 Web 标准。在 Palm Treo 和旧款黑莓等流行的智能手机上，用户经常会因为浏览器的 CSS 支持太差而完全禁用 CSS。我们有时会见到网页使用了复杂而高级的 CSS 并输出给每一台设备。当网页被缺乏 CSS 支持或支持不佳的浏览器不正确地渲染后，看上去可能与预期不符，而且如果网页元素最终被错误摆放或者呈现难以阅读的样式，部分或整张网页就可能变得无法使用。

更糟的是，开发者有时会依靠 JavaScript 来添加或操控 CSS 样式以实现网页功能。这是一个常见的故障点：如果 CSS 或者 JavaScript 其中一个没有得到恰当的支持，渲染后的网页就可能不可用，无论表面之下的标记结构多么干净漂亮都没有意义。

在福特汽车的主页（www.fordvehicles.com）上，全局导航条和底部链接都依赖 JavaScript 将它们摆放到正确的页面位置上。如果没有脚本功能，全局导航不可见，底部导航链接则被放置在主要产品图像的上方，使大部分网页区域难以阅读和无法使用。

耐克的国际登陆页面（Nike.com）有一张使用 JavaScript 和 Flash 的国家列表，用于引导顾客进入某个特定国家的网站。脚本功能被禁用时，顾客看到的只有一张黑色的空白页面，导致他们完全无法购物。讽刺的是，这张网页的 HTML 源代码里有着完全可用的各个国家链接，但却用硬编码的 CSS 隐藏起来，以实现他们认为每个人都能看到的 JavaScript 体验。这个简单的决策每天都会让耐克失去不计其数的潜在顾客。

插件黑盒

Flash、Silverlight 和 Java 这样的插件在开发者和许多用户中很流行，它们能提供强大的功能（比如可视化交互图表、地图和其他信息）、播放媒体文件、连接电脑里的文件系统等。但因为它们能要求用户进行手动安装和升级，而且并不是在每一种平台和浏览器上都能得到支持，所以那些没有安装最新版本的用户经常会被烦人的“升级你的系统”消息所阻挡。

移动设备上的插件支持目前还很零散，因为移动运营商、手机制造商和插件开发者需要在性能和移动设备较差的处理器之间寻求平衡。举个例子，苹果公司坚定地拒绝在 iPhone 的浏览器里支持 Adobe Flash，这就意味着数以千万计的 iPhone 用户目前无法访问任何形式的 Flash 内容。

除了考虑移动访问能力，任何以插件形式传输的内容都储存在插件自身私有的代码结构内，所以无论插件作者如何努力提升可访问性，内容始终无法像用简单语义化 HTML 编码那样易于被屏幕阅读器或搜索引擎访问到。

最后，还有一个虽小却令人担忧的趋势：Adobe Flash 和 PDF 正在被当做最流行的恶意攻击媒介，因为它们在绝大多数浏览器和平台上都得到了广泛支持。这种安全风险的浮现可能会同时影响用户使用率和企业安全标准，继而影响用户访问基于插件内容的意愿。（来源：“Adobe Flash’s security woes: How to protect yourself”，*Computerworld*，12/14/09，http://t.cn/zRlQGFD，以及“Adobe to patch zero-day Reader, Acrobat hole”，*CNET news*，12/16/09，http://news.cnet.com/8301-1009_3-10416816-83.html。）

新设备带来了出人意料的交互标准

许多流行的手机有着卓越的浏览器，在 CSS 和 JavaScript 功能上可以与最新的桌面版浏览器媲美，但它们带来了与标准桌面电脑差别巨大的用户页面交互方式，这可能会明显影响可用性。

举个例子，支持多点触控手势的触摸屏出现在许多移动设备上，成为新的交互标准。轻触和拖曳两种手势结合起来形成了一套新的交互标准，让用户可以缩放、点击和滚动页面。但是，有一些重要的交互方式是 iPhone 或 Android 手机用户在多点触摸环境下无法轻易复制的。

- 在多点触摸设备上，拖曳一根手指扫过屏幕的手势用于平移或滚动网页中的可视区域。既然拖曳手势已经用于平移屏幕，就没有任何手势能让用户在屏幕上拖放物体了。如果你设计了某种交互体验，让用户只能用拖动的方式将项目添加到购物车中，这个功能就无法在这些高级移动设备上实现。
- 一些网络应用程序模拟桌面电脑上按住键盘上的Control或Shift键同时用鼠标点击项目进行多选。在触摸屏上，任何需要同时点击键盘和鼠标的交互方式都不受支持，除非做一点针对性的变通。
- 如果网站通过设置CSS 溢出或用分帧将页面分成多个小尺寸滚动面板，比如某个电子邮件阅读器像Outlook那样有一个列表视图和一个阅读面板，这样的网站在iPhone上无法使用，因为网页浏览器里没有固定显示的滚动条。事实上，iPhone可以完美渲染出整个界面，

但是邮件列表或详细内容面板里的滚动条却不会显示，用户必须凭直觉猜测他们看到的只是部分列表或邮件内容，知道要用两根手指的拖动手势在子面板里进行滚动。

许多网站开发者假定每个能够渲染出增强体验的用户代理，都有完整支持这种体验所需的所有功能和工具，还有一个桌面浏览器以及用于输入的键盘和鼠标。但我们希望你已经知道，做这些假定的风险越来越大。

当企业和个人推出新网站，或在现有系统中加入新功能时，他们面对着如何设计和实现前端代码的重要决策。因为互联网是一种与生俱来无法预测的环境，所以对浏览器和插件做出假定只会营造出一种虚假的安全感。现实是，在网页中使用的任意技术如果不是简单的 HTML 标记，就会在某些场合出现问题，并可能破坏用户体验。

要避免这些陷阱，任何对网站的核心目标或业务至关重要的内容或事务，都应该可以在理解基本 HTML 的浏览器上正常工作，这是互联网底层设计的基石，也是普遍可访问性的关键目标。CSS、JavaScript、cookies 和图像都可以增强某些浏览器体验，也能抑制或破坏另外一些体验。如果一种设计方法能让它们变得次要和可选，那么这种方法就能将最佳体验带给最广大的用户群体。

渐进增强是个好方法

面对日常项目中种类繁杂却都需要我们提供支持的用户和设备时，我们开始思考该如何设计任何人都能用的网站和应用程序，它既能为有条件的用户展现最高级的界面元素，又能在开发和测试真实客户项目时仍然具备可控性。我们需要一种实际的方法来为所有设备提供稳定可用的基准体验，将需要 JavaScript、CSS 和插件紧密配合的富功能设计留给现代的桌面和移动用户。

我们真正想要的是一颗"魔术子弹"：用最简洁、最有效率、向前兼容的开发方式创建一个单一、可控、能到处运行的代码库。

我们相信已经找到了一种近乎完美的解决方案：渐进增强。这个概念最初由 Steven Champeon 和 Nick Finck 在 2003 年 3 月的 South by Southwest 大会上提出（见"Inclusive Web Design for the Future"，Steven Champeon 和 Nick Finck 于 2003 年 3 月 11 日在 SXSW 上的发言，http://hesketh.com/publications/inclusive_web_design_for_the_future）。渐进增强不仅是一种方法论，也是一种思维方式。它首要关注内容和功能。它的目标是只用 HTML 语义的表达潜力来制作具有完美功能性和可用性的网页，创建从一开始就可以访问的页面，在任何能上网的设备（手机、游戏系统、网络冰箱和任何你能想到的东西）上都保证能用。

在 HTML 标记的描述性尽可能增强并且尽可能清晰之后，我们才会开发 CSS 和 JavaScript，两者都写在外部文件里，然后以不喧宾夺主的方式应用到 HTML 标记中，将其转变成一种丰富多彩的交互体验。只要有可能，我们会避免内联样式（inline style）和事件处理函数。渐进增强的关键从仔细区分内容（HTML）、外观与样式（CSS）和行为（JavaScript）开始，这让我们能够为每一种浏览器提供可行和合适的体验。

渐进增强和优雅降级是不是一回事

人们经常会把渐进增强和名为优雅降级（graceful degradation）的编程手法相提并论，有时还会混淆。虽然两者的目标是一致的（确保用户得到可用的体验），但它们的方法却有着根本区别。

优雅降级也被称为“容错”，基本原则是认定复杂系统中的某些部分会出错，然后力求内置一些替代途径来“降级”到一种较差但仍然可用的体验。优雅降级和容错的那些原则早在网页设计出现之前就存在了。事实上，优雅降级背后的理念经常会引用一些网络作为例子，比如如何在出故障的电网或其他类似的复杂基础设施上调整负载平衡。一个常见的互联网优雅降级的例子是noscript标签，这个HTML元素是一种将内容限定传输给不支持JavaScript用户的方法，是正常体验不可用时的备用方案。

渐进增强的核心开发原则采取了一种完全不同的方法。它假定所有基于互联网的系统都能被缩减成一种简单可用的体验，用单一的代码库就能实现，而且在任何地方都能“正常工作”。也就是说，它能将一种可用、可读、功能完整和令人满意的体验提供给最广泛的设备、浏览器和平台。在以最简单的方式满足众人的需求之后，再渐进地（于是就有了名字里的“渐进”二字）叠加上现代浏览器能处理的健壮增强设计，以打造出完整、复杂的体验。

很多年来，我们一直使用渐进增强作为开发流程的基石，并且意识到它是一种当前实际可行而且向前兼容的网站开发方法。

- 它可以实现“普遍的可访问性”，不仅能向屏幕阅读器和其他辅助技术提供广泛的可访问性，还能照顾到禁用了JavaScript或CSS的用户，以及过时或功能有限的浏览器。
- 它促进了代码清晰化：从底层开始思考能使代码更干净、更模块化，每一个功能组件都有唯一的目的，可以在多个上下文界面中重用。
- 它让事物保持集中和简单，能让企业团体只需维护单个统一的代码库，从而兼容所有的桌面、移动和视频游戏设备。
- 它使网站具备向前兼容的能力：今天能正常工作的最简单版本明天继续可用，根据处理能力不同所纳入的功能可以很容易地进行修改，无需进行大修或移除复杂的补丁。
- 它可以实现更简单的后端接口。我们总是使用内建的、完全可用的网页元素作为与服务器之间的单一数据连接，然后用脚本来创建代理连接，使增强的自定义网页元素与基本网页元素之间保持同步。
- 它使得多种体验可以共享一个单一的公用代码库。我们开发的每个网站都能将同一张HTML页面同时用于基本体验和增强体验，因为两者唯一的区别在于CSS和JavaScript如何叠加到这个基础标记之上。

过去的几年间，渐进增强的方法已经悄悄被互联网上一些最大最好的网站所采用，原因就在于它能让这些网站覆盖最广泛的潜在用户。如果你在禁用JavaScript或CSS的情况下浏览Google、Facebook、亚马逊或Digg，你会吃惊地发现这些网站运行状况良好。每个网站实施渐进增强的方

式可能会稍有不同，但每个网站都实现了向任何一位访问者提供可用体验的目标。

大多数情况下，实施渐进增强并兑现普遍可访问的承诺并不需要多做工作，基本上就是做到以下几点：忘记过去的一些坏习惯，换一个角度看待设计和开发，以及确保许多需要做好的细枝末节真正地做好。

本书的目的是通过运用在现实世界中测试过的、简单可行的渐进增强技巧来帮助所有人开发普遍可访问的网站。

第一部分会回顾我们实施渐进增强的方法，包括以一种不同的方式思考如何规划设计，以及用我们独特的方法在应用增强之前测试每种浏览器的能力。然后，回顾 HTML、CSS 和 JavaScript 的最佳实践，它们会赋予你所有必要的工具，支持你以一种更新更好的方式建造网站。

在此之后，我们会分析一些特定的界面组件或部件，一步步地向你展示使用渐进增强方法时会用到的特定标记、样式、脚本和增强辅助功能。这些例子中的每一个都提供了特定的操作指南，合在一起则展示了一套完整的原则和方法，可以扩展到其他编程环境，将渐进增强的方法广泛应用于所有情况。

我们已经在自己的网站和一些客户合约中使用了这些技巧，发现它有助于使渐进增强更加切实可行。希望你读完这本书时也同意这个观点。

目　录

Part 1

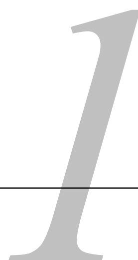
第一部分

测试驱动的渐进增强方法

本部分内容

第 1 章

我们的方法



在高级CSS、客户端JavaScript和Ajax，以及Flash等浏览器插件的支持下，最新的网页技术创新赋予了现代网站视觉吸引力和丰富的交互能力。不过它们有很大的局限性：浏览器和设备对这些技术的支持程度参差不齐。虽然现代浏览器和最新的移动设备有能力渲染出非常复杂的界面，但它们所占的比例还不够大。正如引言中谈到的那样，制作一个只适用于少数高端浏览器和设备的网站或应用程序很难服务于最广大的用户。

我们想要确保客户的内容、消息和功能可以到达每一个人，不仅是那些使用现代浏览器（支持最新网页技术）的人，而是任何有可上网设备的用户。因此，我们在几年前开始将渐进增强的理念引入客户的项目中。

渐进增强的原理很简单：输出符合标准的纯HTML页面，使所有设备都更有可能渲染出可用的内容。然后，只为那些能理解CSS和JavaScript的浏览器在页面上无缝叠加增强的样式和脚本。

不过，开始用这种方法构建网站并测试结果时，我们有了一个重大发现：这种方法没有考虑到许多旧版浏览器和较新的移动浏览器仅部分支持JavaScript和CSS。此外，用户还可能会基于速度、安全或可用性等原因有意禁用这些技术。在现实世界里，CSS和JavaScript必须协同工作才能实现复杂的应用程序界面和组件。（在启用JavaScript但不能正常支持CSS定位的浏览器中，日历组件或滑块肯定不能用。）

在对基于渐进增强的网站进行测试时，我们发现不少浏览器会把一个本来可用的HTML页面“增强”为一团“乱麻”。原因就是这些浏览器并不完全支持它们运行的脚本和应用的样式。然而，我们如何知道哪些浏览器有能力正确渲染出这些增强信息呢？

我们意识到，要实现渐进增强方法，让每个人都有可用体验的目标，需要做三件事。

- 仔细检查设计，确保每一部分（即使是最新潮的Web 2.0或Ajax组件）都是基于结构清晰的语义化HTML，在任何完全不支持CSS或JavaScript的浏览器上，都能提供功能完整的基本体验。
- 在添加增强信息之前，先测试某种浏览器的CSS和JavaScript支持程度，以更好判断是为它提供基本体验，还是应该进行增强。
- 对那些已升级到增强体验的浏览器，要确保花大力气维护可访问性，比如提供键盘导航支持和添加支持屏幕阅读器的功能。

本章将讨论为确定哪些浏览器应该获得增强体验而开发的浏览器能力测试模型，以及它可以

测试哪些功能。然后将介绍我们在日常客户项目中使用的渐进增强方法，这个方法的第一步叫做“X光透视”。这一步会分析某种复杂的界面设计，拟出能支持基本功能体验的语义化HTML，然后制订计划，为有能力的浏览器开发高级CSS和JavaScript，以创造增强体验，同时保持对屏幕阅读器的完整可访问性。

1.1 测试浏览器能力

我们对渐进增强的初始研究揭示出大多数开发者选择的页面增强方式是以下两者之一：向所有启用JavaScript的浏览器传递增强信息，或通过浏览器嗅探（检测Internet Explorer等特定用户代理或WebKit这样的渲染引擎）仅对特定的一组浏览器输出增强信息。

我们从一开始就排除了浏览器嗅探，原因有下面几点。

- 有效的嗅探需要精确了解每种浏览器的行为（以及其各个版本的变化情况），这使维护脚本成为一项巨大、复杂和永无止境的挑战。
- 根据定义，它不是向后兼容的。你只能嗅探当前存在的浏览器，如果某种能提供增强体验的新款浏览器明天发布，它就会被拒之门外，除非把它们加入列表。
- 即使是最详尽的用户代理白名单也可能会失效，因为浏览器可以被配置成报告错误的用户代理信息，从而绕过这些技术（也被称作浏览器欺骗）。

所以第一种方式（向所有启用JavaScript的浏览器传递增强信息）看上去是更好的选择，因为大多数支持JavaScript的浏览器都能够渲染出增强的体验。但是前面提到，某些浏览器只是部分支持这些增强信息，从而导致布局混乱和JavaScript错误，这种情况的数量多得惊人。

观察这些情况时，我们注意到这些缺损的体验植根于两大模式：一些浏览器错误地渲染了CSS，原因是它们不能很好地支持盒模型（box model）、定位、浮动或其他源于网页标准的常见CSS属性；另一些则不能很好地运行常见JavaScript功能，例如DOM操作、事件处理、调整窗口大小和执行Ajax请求。

如果我们有一种可靠的方法能检验若干有恰当代表性的浏览器能力，然后仅在我们对CSS和JavaScript能正确协同工作有十足把握时才输出增强信息，那么应用我们的渐进增强方法就会容易和可靠得多。

带着这个目标，我们通过不断试错开发出了浏览器能力测试框架。其中的JavaScript基本检验项目相当简单：我们的测试使用了一种叫做对象探测（object detection）的方法，通过它实际上能询问浏览器是否识别如`document.getElementById`函数这样的本地对象，并得到明确的`true`或`false`回应。每项测试都以一种容错和非干扰性的方式编写，因此如果某种浏览器不理解一个JavaScript方法，它不会报错。

更具挑战性的问题是如何判断某种浏览器是否能正确支持高级CSS技术。没有任何原生方法能使用对象探测来询问某种浏览器是否能恰当渲染具体的CSS特性，例如浮动、定位、垂直分层元素或透明度。

因此，我们设计了一套CSS能力测试用例专门做这件事。每一项CSS测试都使用JavaScript，

将不可见的HTML元素插入到网页中，应用一组特定的高级CSS规则，然后运行一个JavaScript函数来测量结果。举个例子，要了解浏览器是否正确支持定位，测试会使用CSS将一个div放置在某个特定位置，然后运行一个JavaScript函数来比较测出的坐标是否与参照条件匹配。

一种浏览器通过全套能力测试后，我们就可以很有把握地认定它对CSS和JavaScript两者的处理都是一致并且符合标准的，应该能够良好地渲染出增强体验。到这一步，测试会动态加载高级样式表和脚本，将基本的HTML标记转变成增强体验，并在页面中添加一个链接以供那些偏爱简单版本的用户关闭增强体验。最后，测试会设置一个cookie防止之后再次运行能力测试，从而提升页面速度。

将测试框架实际投入使用时，我们见到了它为渐进增强项目带来的实际益处。首先，它给了我们一种非常可靠的方式来划分哪些浏览器可以正确显示增强体验，哪些不能，因此可以大幅降低本来可用的基本页面被增强信息损坏的危险。又因为这个测试只会在那些通过的浏览器上加载增强版CSS和JavaScript文件，所以它让我们能提供小得多的、更加流线化的基本体验（不会预先载入大量标记、样式或脚本），结果是下载时间大大加快，不必要的服务器请求也变少了。

这个测试框架的设计是高度灵活和模块化的，可以自定义修改，以测试具体客户项目要求的特定CSS和JavaScript能力。如果不存在复杂的CSS浮动或Ajax脚本，我们就可以从测试标准里移除相应的代码。

在我们的项目中，通常会运行一套能力测试，简单地将浏览器划分成“基本”或“增强”两个组，以方便编程和维护。我们将这种对基本和增强的区分视作以最小的力气为最广泛的设备提供访问的方式。从这种基准支持程度开始，你就可以相当容易地进一步定制测试脚本，将浏览器更加精细地划分成多个能力等级或为特定设备（比如iPhone或Kindle）提供优化过的体验。

第6章将详细分析这个能力测试的结构和运行机制，并讨论一些能体现模块化方式优点的情况。

1.2　规划渐进增强：X 光透视

随着用一轮轮的迭代对能力测试进行打磨，我们开始发展出一套方法，将复杂的网页界面设计进行分解，使之适合开发渐进增强和能力测试。

有时，找出能正确支持某种高级设计组件的原生HTML元素非常容易：一个自定义样式下拉菜单的外观和行为非常像原生的select元素，不用多想就知道该从它起步。类似地，自定义样式的复选框可能会存在一些打造样式方面的难题，但是我们从一开始就知道它们是复选框。

然而，其他一些案例却没有这么明显：一个Netflix样式的、Ajax驱动的星级评定组件应该使用哪种基本标记？我们在众多新闻网站都能看到的“最热门/最多邮件发送/最多评论”标签式内容组件又该如何处理？那些能在Kayak和其他电子商务网站上看到的，用于过滤结果的自定义日期或价格范围滑块呢？那些更加复杂但却日益流行，类似于Gmail这样使用拖放等富交互手段的Ajax驱动型应用程序呢？你无法从通常受支持的HTML元素工具箱里找到符合这些情形的精确匹配。

话又说回来，虽然这些例子都是高度定制的并且交互性丰富，但它们支持的用户操作（在刻度栏上选择一个选项，从一个内容区域切换到另一个，设置某个范围的低点和高点，排序，搜索和检索）却明显能用标准HTML做到。我们只需一些创造性思考，解构这些组件和交互方式，然后确定用哪些原生HTML元素能完成同样的工作。

主要挑战在于，如何透过自定义样式、动画和其他行为看到隐含的基本运动部件。我们把这一过程比喻成"将组件放到X光下"：举个例子，那些整洁的CSS和脚本功能使自定义滑块具备交互功能，但它们其实只是皮肤和衣服，在基本体验中，滑块的"骨骼"则是用来设置高低数值的文本框输入，或选择一小组选项的单选按钮，甚至还可以是用来选择一大组选项的选择菜单。

X光透视在审视复杂的设计时会变得更加复杂和有趣，比如组合多种内容类型和交互组件的网站，使用动态内容和复杂布局的网络应用程序，或者根据用户交互情况在页面中选择性提供内容的工作流程等。

在宏观层面上也有一次X光透视过程，以识别网页各主要部分（或多个网站页面之间的共有模式）如何组合在一起，寻找可能揭示出内容和功能关键部分组合关系的行为模式，并评估如何优化这些资源的序列，以确保它们在基本和增强环境下都能正常工作。

进行这种高阶分析时，组件或元素层级的基本原则同样适用，即确定服务用户所需的所有关键内容和功能，考虑在某种特定情形下可以使用的HTML元素（基于内容格式、数据需求、业务规则或整体流程），确定能够提供最佳用户体验的标准HTML标记，很简单。

第2章会更详细地讨论X光透视，并考虑一些能探究解构过程细节的复杂设计方案。此外，第8章~第19章会把界面组件一个个地放在X光下，看看渐进增强技术如何让基本和增强体验都尽可能具有完整可访问性、功能性和可用性。

在深入探讨如何将X光透视实际应用于网站和组件之前，我们先介绍一下在项目中成功进行渐进增强开发的流程。

1.3 从 X 光到实践：渐进增强开发的构成

随着我们不断打磨能力测试框架并应用X光透视，一套术语和方法开始浮现，定义了我们的渐进增强开发过程。我们需要支持两种体验："基本"体验对所有能上网的设备（在我们力所能及的范围内）全部有效，另一种"增强"体验则用于功能更强的浏览器。前者的标记是构建其他一切事物的基础，所以我们就将它命名为"基础标记"（foundation markup）。又因为功能更加丰富的体验依赖于高级的表现方式和行为，所以我们称呼任何实现它们的标记、样式表或脚本为"高级"（advanced）或"增强"（enhanced）资源。

为了成功实现这些体验，必须遵守渐进增强方法的三条关键原则：

- 以清晰的内容和恰当的标记作为起点；
- 坚持布局和表现严格分离；
- 高级行为和样式层无缝叠加，兼顾可访问性。

鉴于我们为基本体验设定的目标是普遍访问，所以开发方法必须以清晰的内容和恰当的标记

作为起点。排列整齐的语义标记是基础，它既有意义又具备功能性，在此之上可以添加增强信息；它也更有可能在更广泛的设备上具备可用性；它还为用户使用辅助技术导航网站提供了一张清晰的地图。

围绕网页内容选择标记的方式会大大影响CSS和JavaScript等增强信息的加入方式，以及你的网页内容对那些视障用户的可访问性。一个由干净、整齐、结构良好且准确的标记构建而成的网站能够大大方便制作样式和交互，还可以重用代码。

为了给基本和增强体验打造出最健壮的基础标记，关键在于理解语义化HTML的特性、功能和限制，以及当前HTML官方规范里的一系列元素、标签和属性（乃至后续规范中放置向前兼容代码的新功能）。第3章将分析这些方面，并推荐最佳方法。

我们在渐进增强开发里严格遵循的第二个关键原则是*布局和表现严格分离*。首先制作出网页的线框图（wireframe），然后再填充内容，因为这样大大简化了项目级模板系统的创建工作，同时将所有受CSS影响的表现和样式与内容严格分离。

当一张网页的结构和样式被构建成独立于它的内容时，就能很容易地创建相同布局的多种变体，为各种各样的浏览器和设备提供最佳体验，而不用使用会影响内容样式的结构性CSS。然后，就有了考虑更多媒体类型（标准桌面屏幕、移动设备、印刷品和辅助设备）的灵活性，有选择地运用基本和高级CSS特性来适应它们。第4章将分析哪些类型的样式能安全地应用于大多数环境，那些相对更复杂的CSS规则是如何工作和互相影响的，以及如何为最简洁的基本体验和最健壮的增强体验集中并优化样式。

用JavaScript实现的高级行为和表现能大大增强用户体验，但是如果用得不合适或者不当心，就会导致一大部分用户完全无法使用某个组件、网页甚至整个网站。许多优秀的指南和明确的最佳实践方法都介绍了如何组织和引用脚本，从而*无缝叠加增强行为*，让它们能够安全地提升有能力的浏览器的体验，同时也能确保基本体验不受损。第5章会详细说明这些原则和技巧，以及相关的可访问性考量。

在更好地理解HTML标记、良好定义的CSS以及非干扰性JavaScript如何协同工作之后，第6章将深入介绍能力测试，看看如何使用上述原则和方法来实现一种更为可靠的渐进增强体验。

1.4　理论结合实践

接下来几章会重点介绍最佳实践，帮助你理解如何在真实的项目中成功实施渐进增强理论。下一章开始先介绍如何使用X光方法，之后几章会涉及编写有意义的HTML标记、有效应用CSS，以及使用JavaScript来制作增强和添加行为。最后，第6章会详细讨论浏览器能力测试套件。

第2章 渐进增强实践：X光透视

我们构建网站的首要目标是根据每个人所用浏览器的能力和局限，尽可能给他们创造最佳的体验。我们相信通过渐进增强的方法就有可能实现这个目标，它为那些使用现代浏览器的用户提供了高级代码带来的所有益处，同时也能满足使用旧式浏览器、屏幕阅读器和可上网手机用户的需求，而这一切都基于一个统一的代码库。

通过在许多客户项目上的不断摸索，我们设计出了一套基于渐进增强的自定义方法，它有两个重点：一是能力测试，用来明确地判断每种浏览器具体的能力；二是我们称之为“X光透视”的设计与开发规划流程。

X光透视基于这一原则：即使是最复杂的现代网页设计（比如那些有着类似桌面程序行为、基于Ajax的动态应用程序）也可以并应该能表述成简单的语义化HTML，从而为所有访问者提供一种可用且可访问的体验。使用X光透视的意思是，“透过”一种设计中复杂的组件和视觉样式，分辨出组成网页的核心内容和功能要素，并找到等价于两者的简单通用HTML。它的意思还包括了明确各个组件的复杂样式和行为，然后制定一套策略，视情况将这些增强应用到有能力的浏览器上，同时保持可访问性。

X光流程是一种强大的工具，能让你有策略地思考添加增强信息的最佳可行方式。它能让处处死路的恼人体验变成完全可访问的顺畅体验。你要做的仅仅是“忘掉”某些常见的错误编程习惯，培养新的策略性思考方式：首先，尽可能建立起最完整的网页基本体验，然后，再把那些很酷的内容叠加上去。

这一章会概述我们使用的X光方法的原理，然后回顾4个真实的设计案例，里面包括了一系列高级界面元素和交互方式。这些案例研究会展示如何用X光透视评估一种复杂的设计，回顾我们在制作基础和增强标记、CSS和JavaScript脚本时所做的开发和编程决策，还讨论了为确保广泛可访问性而添加的一些额外增强信息。最后，列出了测试代码时需要考虑的标准要点核对表，以此检验所有围绕渐进增强付出的努力是否的确能给所有用户都带来满意的体验。

2.1 X 光透视概述

X光透视是我们开发的一套方法，用来评估复杂的网站设计，将其分解成最基本的组成部分，然后再使用某种方式组装起来，使包含同样代码的网页既能在功能齐全的现代浏览器上工作，也

能在所有其他浏览器和设备上工作，哪怕它们除了理解基本的HTML外什么都不会。

X光的流程始于一份目标设计，里面展示了最终页面在现代浏览器上应该具备的外观和行为，包括所有的高级装饰元素。带着这份目标设计，我们进入规划和开发流程，它由三个关键部分组成。

(1) 定义整体的内容层级和优先级，找出各个组件对应的基本HTML元素。

(2) 制作能提供所有关键内容和功能的“基础”标记。此标记尽可能少地使用“安全”样式，不使用JavaScript。

(3) 编写高级标记、CSS和JavaScript，为有能力提供支持的浏览器添加视觉和功能增强。

用代码构建基础和增强体验通常是一个不断重复的过程。当我们编写完基础标记的初稿，开始研究在哪里添加增强信息时，通常会发现在某些情况下为了实现这些增强，必须对基础标记做一些细微调整。

2.1.1　定义内容层级并将组件映射到HTML

第一步的工作是建立优先顺序，既自上而下地辨识出整体的内容分组和层级，同时又自下而上地定义出与各个内容和功能等同的基本HTML。

关于内容分组和层级，你需要考虑几个问题。

- 页面哪些内容最重要？要理解网页传递的目的或者消息，用户们需要阅读/注意什么？内容是否存在明显或者隐含的次序？
- 纵观整个网站设计，是否存在可以且应该在模板系统里重复使用的公共内容、功能或行为模式？
- 是否存在必须完成的任务？如果存在，那么整个过程包含了哪些步骤，用户会用到哪些工具？这些步骤是必须的还是可选的？这些步骤里的内容和选项是否依赖于用户之前的选择？

这些高层次的问题让我们有机会建立起界面规则和标准，成为集中化样式的基础和编码过程中的功能行为规则。

在进行高级分析的同时，我们还仔细地观察网页内容和功能的细节，以及它在基本和增强体验中的工作原理。

- 该设计是否需要用Ajax动态插入内容？（在基础标记里，内容必须随网页一同输出，或者包含在一张单独的HTML链接页面里。）
- 界面里是否有某些部分会构成一种工作流，使某一步中做出的选择决定另一步中的选项，或者要求传输细节信息到一台服务器来验证某个选择？我们需要确保在基本体验里隔离所有意外因素，以帮助用户实现效率最大化，并尽可能减少错误和麻烦。
- 界面中的某些部分是否会过度占用带宽，或者在由标准HTML构成的基本体验里难以使用？如果是，我们可以在基础标记里提供简化的组件，也可以鼓励初级用户使用某种线下的替代方式来完成目标。

最后，我们详细分析页面中的每一个组件，辨识出它的用途，然后确定能对其提供最佳支持的基本HTML元素。同时，我们会思考可以如何将CSS和JavaScript增强信息应用到这个标记上，并评估这些增强信息是否会引入任何具体的可访问性问题。

我们自己在工作中经常会卷入到有趣的争论中，因为我们试图搞清楚该如何用有限的HTML语句来表达非常复杂的用户界面。一些基本的HTML决策是很直观的，比如简单的页面标题和正文内容很明显应该格式化为标题和段落元素。但是其他一些（特别是那些复杂且与标准HTML元素没有明显对应关系的交互组件）有时需要更仔细地思考。

举个例子，考虑一下这个过滤航班时间和价格的自定义滑块控件，如图2-1所示。

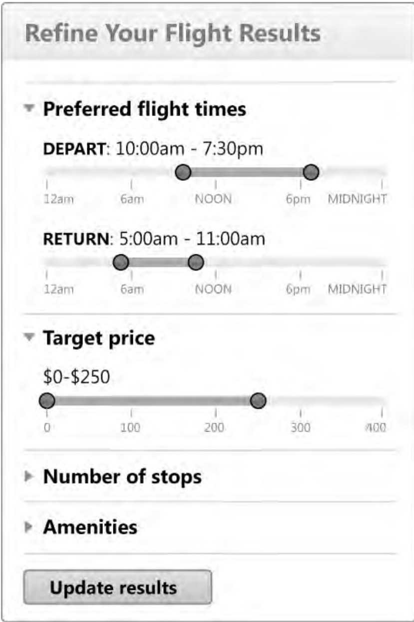


图2-1　在一个旅行搜索网站里用作过滤器的滑块

怎样的HTML能够正确对应这个用于设置起止时间段的滑块？或者对应设置美元价格上下限的滑块？两者是否相同？

这时就能用到X光透视法了。我们将注意力集中在元素应该传递什么信息，或者需要从用户这里收集什么数据上，而不是观察元素在最终设计里的外观或者用户交互方式。Kayak的这些滑块的真正作用是什么？

考虑到根据设计，每个时间段滑块会收集两个值（一个开始时间和一个结束时间），范围在24小时之内。有几个收集用户输入的标准HTML元素可以很有效地捕捉到那些值，包括文本框或者下拉列表框元素，但是哪一种能提供最佳的用户体验？这个时间选择器还有一组固定的连续选项，所以一对下拉列表框元素（里面可以设置以15分钟为间隔的标准时间增量）会是用于基础标记的一个不错选择，因为它提供了类似于滑块刻度标识这样的明确用户限制条件（它只允许选择

有效的时间）。对于增强体验而言，我们会将下拉列表框元素的标记（它们的选项、值和其他有关属性）作为跳板，用CSS和JavaScript创建一个功能完备的滑块组件，将它的值“传送”给原始的下拉列表框元素，这样就可以随表单一起提交了。（第16章会详细讨论将渐进增强应用于可访问滑块的建议。）

通过考虑高层次的内容等级、优先级和分组，再把具体的内容和组件功能映射到等价的基本HTML上，你就有了一张初步的路线图，用来在基本体验里编写简单的标记和尽可能少的样式，然后再加入复杂的标记、样式和脚本作为增强。

2.1.2　编写基础标记和尽可能少的安全样式

当网页的整体层级和每个组件的用途都规划好后，高效编写作为基本和增强体验共同基础的语义化HTML代码就变得容易多了。我们将这种HTML称为“基础”标记，因为它的确是个基础，我们会在它上面添加CSS和JavaScript增强，从而使最终设计变成现实。

编写基础标记时，首先处理大的界面版块（包括所有主要容器元素，比如大标题、页脚和栏目设置）来稳固基本网页结构，然后再处理具体的组件。接下来，为页面里所有的每个元素填充内容和功能，特别注重使用语义化HTML创建出易于使用的基本体验。（第3章会讨论如何编写同时支持基本和增强体验的语义标记。）

在某些情况下基本体验需要使用HTML元素来实现功能，但是增强体验却不需要。在另一些情况下，标记只有在增强体验里才是合适和有用的。举个例子，在某个结果页面的基本体验里，一个被用作过滤器的`select`元素需要一个按钮来提交表单，而这个按钮在增强代码里却不是必须的，因为JavaScript可以自动在“失去焦点”（blur，即用户完成选择后将输入焦点移到页面的其他位置）时提交表单。相反，一个打印按钮则需要JavaScript才能工作，它应该被排除在基础标记之外，这样就不会在基本体验里困扰用户，当页面被增强后可以使用JavaScript再添加进去。

提示　编写基础标记并记录增强信息应该放置在何处时，可以用直接写在标记中的HTML注释来备忘，比如：

```
<!-- 这组单选按钮会被增强为滑块 -->
```

或者

```
<!-- 打印按钮放在这里 -->
```

这种注记能在开发过程中保留重要的增强相关信息，而且还为将来更新设计提供了有用的路标，或者用于在规模更大的协作型设计小组之间交流沟通。

编写基础标记时，我们会定期在众多浏览器和屏幕阅读器中预览页面，检查内容层级是否完好，网页是否能正常阅读，是否有意义，是否能独立正常工作。使用恰当语义编写的结构良好的标记天生就是可访问的：链接和表单控件都可以使用键盘访问，图像上的`alt`文本为屏幕阅读器和搜索引擎添加了文字注解，整齐的源代码顺序使网页易于阅读。

这还是一个验证标记的好时机，对发现标记错误和避免后续漏洞可能会非常有帮助。

提示 我们通常使用的一种验证工具是W3C提供的，网址是：http://validator.w3.org。

如果基础代码已经成形，就可以应用少量的“安全”样式规则来改善基本体验的视觉外观和可用性。安全样式是非常简单的CSS规则，能在众多浏览器里可靠地渲染，即使不被支持也能优雅地降级。

安全样式通过外部样式表应用到基础标记上，可以包括字体序列（font family）和非常有限的一组背景颜色、边框、内边距和外边距，仅能够传递基本的品牌特色以及为内容里的对象创造出舒服的间距，以此强调内容层级，并使网页易于阅读和浏览。

一些CSS属性可能会被旧版或移动浏览器渲染得出人意料，因此避免在基本体验的安全样式表里使用它们至关重要。它们包括浮动、在深色背景或图像上反白文字、设置固定的宽度和高度，溢出（overflow）属性以及外边距或位置的负数值。（第4章会更详细地讨论如何编写安全样式。）

完成基础标记和安全样式之后，网页应该具备了完整的功能并可以被每个人使用。下一步，我们会规划增强体验。

2.1.3 应用标记、样式和脚本增强

编写增强组件（比如自定义滑块控件）时，经常需要调整基础标记，从而更好地实现增强体验。举个例子，我们经常会给基础标记的HTML元素添加ID和类（class），以建立应用样式和行为增强所需的“挂钩”。如果在增强体验里应用多重背景图像或者其他视觉效果需要额外增加标记，还可以看情况选择使用`span`和`div`标签。因为在基本体验的用户眼中这些属性和标签是不可见的，所以可以将这些无害的元素添加进基础代码里，让增强过程能够更容易地制作样式和JavaScript交互。我们照旧还是建议有选择及适度地使用额外标记，因为额外的标签会增加页面体积，过多的额外复杂性和网页体积会对基本体验造成负面影响（增加下载时间）。

编写完标记后，增强的样式会承担大部分的重体力劳动，将内容移到浮动或固定位置的栏目中，插入背景图像，应用让交互组件正常工作所需的样式。就其本身而言，它们可能不会在所有的浏览器上以同样的方式工作，也许需要分割成多个样式表，在能力测试通过后有条件地加以应用。

这时还应该添加JavaScript函数来转换基础标记、操控样式和添加行为，将网页转换成增强体验，并移除只在基本版网站里有用的标记。

另一个关键在于，要确保任何会在增强体验里由Ajax添加或更新的网页内容，都必须以一种可访问的方式传输给基本体验里的用户。举个例子，数据网格、搜索结果页或产品介绍应该由基础标记里的数据生成，而不是构造成空白的`div`，在页面加载后再由Ajax生成。任何更新网页数据的过滤、分页或导航都应该使用标准HTML表单和链接制作，以确保导航在脚本功能未启用的情况下也能工作。这条看似简单的规则经常会被忽视（可能是考虑到了速度或者响应性），但通过本书引言部分的那些例子我们看到，对缺乏完整JavaScript或CSS功能的设备用户来说，输出网

页时遗漏必要内容可能会导致糟糕的体验。

用X光透视来设计和开发网页只需稍多一些的预先规划时间，但是它能带来极大的好处，表现在更干净地区分了标记、表现和行为。最重要的是，它让渐进增强这个目标得以实现。

接下来看看4个真实的设计案例（一个新闻网站，一个零售结账表单，一个动态的预算规划工具以及一个照片管理应用程序），它们包含的高级界面设计体现了渲染或者可访问性的挑战。每个案例都会图解展示如何将一种设计解构成基础标记、样式和脚本增强，并重点强调如何在基本和增强体验里同时保持可访问性。

2.2　案例 1：规划新闻网站的结构和组织方式

近年来，新闻网站已经发生了转变，吸收了大量的增强样式和基于JavaScript的功能，比如动画幻灯片、紧凑的过滤和排序控件、动态图表、地图以及其他丰富的数据显示工具。虽然这些增强功能显著改善了体验，但是它们需要更为复杂的代码结构，并带来了潜在的可访问性挑战，这就让使用渐进增强构建这类网站成了理想选择。

看看新闻网站的目标设计，如图2-2所示。

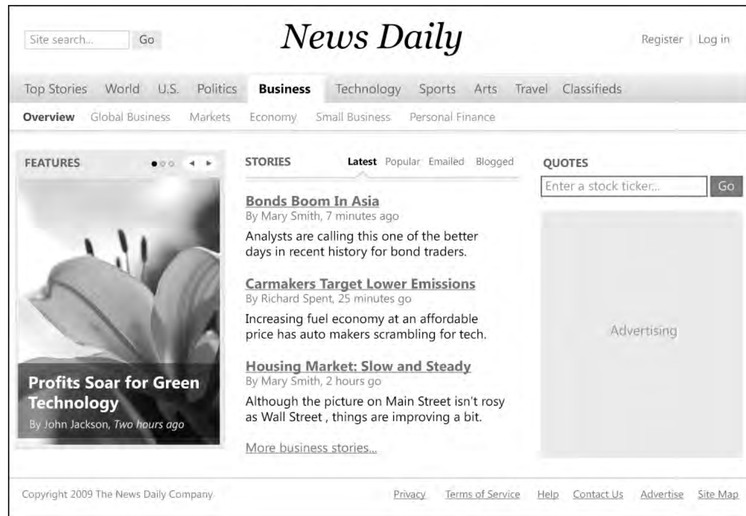


图2-2　带有互动功能的新闻网站目标设计

这个目标设计引入了许多交互组件，包括一个醒目的自动切换精选区，一张带有几个过滤选项的文章链接列表，以及一个自定义的股票报价搜索工具。

2.2.1　评估内容组织和命名方式

首先，我们会评估这个最终设计，理解内容的分组方式，并确定是否有具备高级功能的组件需要被映射为简单的标准HTML。

第一步是组织和命名网页内高级区域。这个布局看上去很直观：从顶部开始是一个刊头，附带搜索框和徽标；然后是一个两级的全局导航条；网页的主要部分是一大块内容区域，分为三栏；页脚则是版权和支持链接。我们会给每一个区域分配一个描述性的ID，简要并清楚地描述它的用途和在网页中的优先级，而不是它的位置或者样式（因为这两者会随着网站的扩版或重新设计而改变）。

为了组织这些区域，我们会将顶栏的ID命名为masthead（刊头），全局导航是navigation（导航），页脚栏是footer（页脚）。为页面主区域中的三栏命名ID则有一点点棘手。比较平常的命名方案是将精选幻灯片栏命名为featured-content（精选内容），中部比较宽的一栏是main-content（主要内容），右手边有搜索和广告的一栏是sidebar（侧边栏）。这种命名惯例是可行的，因为它足以描述栏目的优先级和相互关系。

我们还应该为每个内容区域分配一个WAI-ARIA路标角色（landmark role），如图2-3所示。ARIA是W3C创建的一项规范，为屏幕阅读器在标记里添加额外的语义注释，比如特定组件的角色或活动状态。路标角色可分配给静态内容块以标明它们的主要用途，它们还是屏幕阅读器用户在网页中导航的一种快捷方法，因为许多新版屏幕阅读器都启用了这项功能。路标的分配方式是`role`属性和一个被认可的值，比如`navigation`或`main`。被认可的角色及其描述的完整列表可以在最新的WAI-ARIA规范里找到：www.w3.org/WAI/PF/aria/roles#landmark_roles。

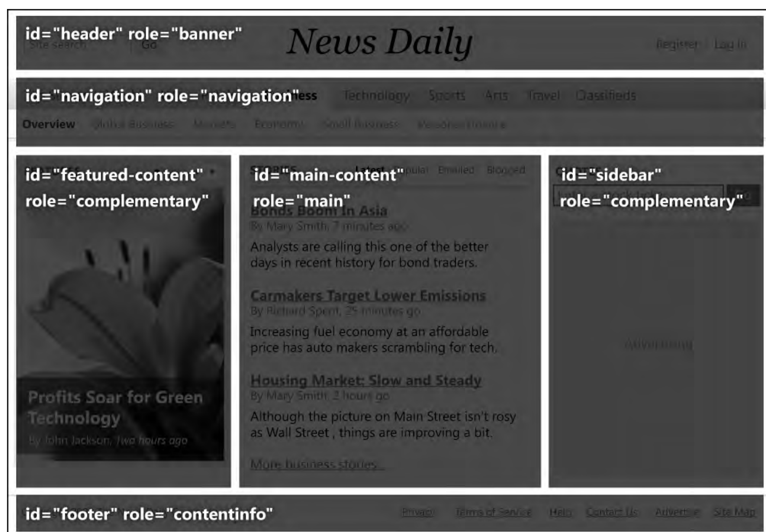


图2-3　这个新闻网站的主要布局块分配到了合乎逻辑的ID名称，并被赋予了路标角色

2.2.2　借助原生HTML层级功能实现内容组织

在基本体验里，你无法使用高级CSS和JavaScript创建多栏布局，或使用幻灯片或标签页来流线化呈现内容。相反，基本体验必须完全依靠源代码顺序和语义标记结构来表示内容组织和优先

级。标题（heading）元素是以可视化方式组织网页内容的最重要工具之一，它的结构里内建了层级，并为搜索引擎和使用屏幕阅读器的用户提供了额外的语义注释。我们在此将借助标题的原生能力提供大体的结构。

编写基础标记时，重要的一点在于分析所有的网页内容，然后勾勒出一套相似、全面和一致的标题结构。下面介绍如何为新闻网页建立标题结构，从而清晰地概括文档组织。

- 刊头（包括刊物的名称《每日新闻》）被标记为一个h1元素，因为它是网页里最突出和最重要的标题。
- 主要内容区分成三栏：精选（Features），文章（Stories）和包括一个股票报价搜索工具和广告的侧边栏。每一栏的顶部标题都被标记为一个h2元素。
- 每一栏里的内容对象，比如幻灯片标题、链接文章标题和市场数据标签都被标记为h3元素。

以这种方式运用的标题元素能帮助一般用户和屏幕阅读器用户辨认出内容层级。默认情况下，浏览器会将标题元素按照字体大小分级渲染，从非常大（h1）到小（h6）。在基本体验里，我们以浏览器默认的字体大小作为标题的样式，这样它们看上去比普通文字更大。

使用标题元素，新闻页面就有了层级和顺序，并且还有其他好处，那就是能帮助那些只依靠页面结构来导航的屏幕阅读器用户。如今，大多数现代屏幕阅读器都提供了某种控制方式，让用户可以顺着网页标题（以及段落和列表）浏览内容。

2.2.3 构建导航

增强体验里的顶部导航是一个紧凑的横条，通过变换选择的一级导航选项的背景颜色来提供状态反馈，并将它与二级导航选项分到一组，如图2-4所示。

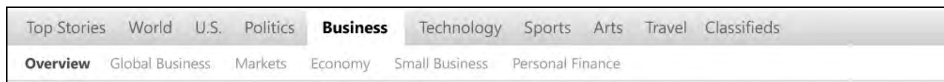


图2-4 新闻网站目标设计里的顶部导航

在基本体验里，这个导航的最佳表现形式是一个无序列表，因为它能体现出选择的一级导航选项与它的次级导航选项列表之间的层级关系。每个列表项都包括一个链接，在基本和增强体验里点击后都会刷新页面。在基本版的网页中，导航链接的无序列表被渲染成一长列嵌套的链接。

当它被放置在源代码顺序的顶部时（仅低于刊头），在移动设备和屏幕阅读器上使用键盘命令进行网页导航的用户，可能不得不滚动或看完这长长的一列选项才能到达更重要的精选内容部分，所以更好的做法是提供一个可选链接，让用户跳过在每个页面上都重复出现的导航元素。你只需附上一个标注为“跳转到内容”（Skip to content）的锚链接（anchor link），如图2-5所示。将它的href值设置成主内容区域的id（本案例中是content）即可：

```
<a href="#content">Skip to content</a>
```



图2-5　用户在基本体验里见到的顶部导航无序列表，上方有“跳转到内容”链接

我们在设计中选择只显示选中项目的次级导航，但是别忘了所有10个顶级新闻分类相关的子菜单可能有50~80个菜单项，或者更多。针对那些有着数以百计选项的复杂或深度嵌套的导航系统，我们有时会在源代码顺序里把导航标记放置在网页内容的下方，这样用户就不会在每次载入新页面时都遇上（然后不得不跳过）整个导航。这样可以在更靠近屏幕顶部的地方显示主要内容，使手机或屏幕阅读器用户能够更容易地进行快速访问。

提示　采用这种替代方法时，为键盘用户提供一个位于顶部的Jump to navigation（跳至导航）链接会很有帮助，能让他们快速访问到底部的导航块。

2.2.4　处理分层和动画内容

页面主内容区域的第一个内容栏里包含一张精选幻灯片，带有动画渐变和暂停/播放控件。这个幻灯片很花哨，但从基本层面看，它其实只是用一种紧凑的方式显示多篇精选文章的链接，每一篇都带有一张图片、一个链接到详细内容页面的文章标题以及一段文字说明。如果用X光透视观察此组件，那么这张幻灯片可以简单地使用一连串内容块组建，然后将它们写入页面成为基础标记的一部分，这样无需JavaScript也能访问，之后再为有能力的浏览器增强成幻灯片。

在基本体验里，幻灯片的所有内容都可以格式化成静态的电影胶片形式，同时显示所有照片和文字说明。通过这种方式，所有内容都可见而且可以访问，我们能很容易地将这组内容块转换成增强的幻灯片体验。

这张幻灯片的控件（暂停/播放、上一页和下一页等控件用于在多张幻灯片中导航）只在有脚本和CSS增强的体验里才是必需的，在基础标记里完全可以省略，当能力测试通过后再用JavaScript插进去，如图2-6所示。



图2-6　增强体验里的动画幻灯片基于基本体验里一连串具备完整可访问性的内容块

某些情况下，在基础标记里输出所有精选内容可能太占带宽。如果事实的确如此，另一种可行的方法是让基础代码只包括一篇精选文章，以此减少网页体积，然后再为增强体验里的用户用Ajax将其余文章标记载入页面。

2.2.5　支持动态过滤和排序

这张网页在精选块的旁边显示了一排文章列表，用户可以过滤最新（latest）、最热门（most popular）、邮寄最多（most emailed）或者最受博客作者关注（most blogged）的文章，如图2-7所示。

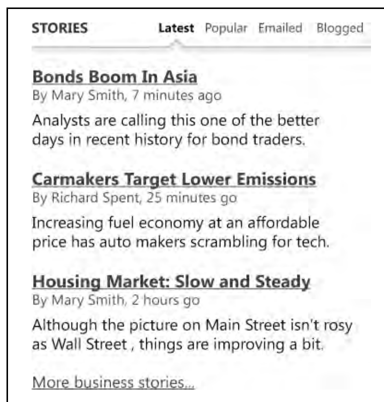


图2-7　在增强体验里，过滤链接使用Ajax动态载入新的文章

通过X光透视我们发现，如果这个页面一开始基于默认过滤选项（“最新”）加载一组初始文章，并将过滤控件制作成标准链接，点击后用替代组文章刷新页面，那么这项功能对基本体验里的所有人来说都是可访问的，如图2-8所示。

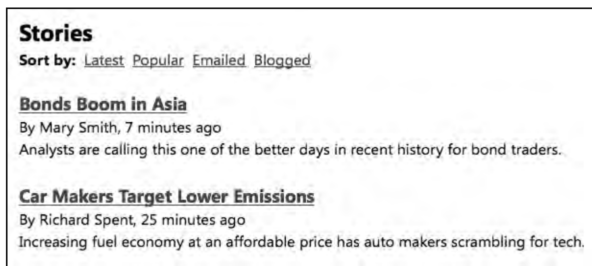


图2-8 在基本体验里，过滤文章视图通过普通链接进行切换

在基础标记里，每个过滤链接的href属性都会包括一些参数，用过滤之后的文章列表来刷新网页。在增强版本里，可以使用JavaScript来截获点击某个链接的操作，解析链接的href值，然后用Ajax动态重新生成文章列表，而不是请求刷新网页。

这项功能还可以用带有单选按钮的小型表单实现，或者用一个包含排序选项的select控件，再加上一个“发送”按钮来提交表单。至于如何选择在基本标记里实现像这样的特定交互方式，取决于服务器的设置，以及感觉上哪种方式最可用、最适合目标设计。

2.3 案例 2：结账表单中的工作流、验证和数据提交

请思考一种通常会采用增强动态功能的在线案例：电子商务结账表单。

许多当代的结账页面使用由JavaScript、Ajax和高级CSS控制的功能来实现高效、愉快的交易，将出错的机会降至最低，从而大大增强用户的信心。举个例子，结账页面通常包含这些功能：表单元素根据用户输入渐进地显示出必填字段；对表单元素进行即时数据验证、高亮和错误反馈；在表单字段边上甚至内部显示帮助文字和操作指南等提示；滑块和日历等高效的数据输入组件。这些功能从视觉上对表单进行了组织，将焦点和信息放在用户最需要的地方，并让表单变得更吸引人和易于使用。

缺点在于，这些功能离开JavaScript和CSS后会变得不好用，甚至完全失效，可能导致一部分用户因为浏览器太旧、系统太慢或者使用手机而被拒之门外。大多数在线销售企业花费了大量的时间和努力说服人们在他们的网站上购物，如果因为开发者决定走捷径而导致哪怕只有一名顾客止步于最终的结账页，都是不可接受的。

那么，观察一种包含某些高级功能的结账页设计，看看如何设计方案，从而在基础标记里提供所有必需的内容和功能，并为向有能力的现代浏览器添加增强功能而铺平道路。

2.3.1 解构结账表单设计

我们的目标设计是一种紧凑的结账工作流，将数个单独的步骤组合到一个页面中，如图2-9所示。

图2-9　一种高级结账交易页的目标设计

制作可访问结账表单的第一步是站在高处审视目标设计，找出所有无法精确映射到标准HTML的功能或组件。不少高级功能立即映入我们的眼帘。

- 多列布局、设计丰富的表单区域。
- 有着独特品牌视觉风格的自定义样式下拉列表框，以及在礼品选项和信用卡选择控件里用来提供反馈的缩略图。
- 一组由单选按钮实现并以标签形式工作的支付选项，每一项都显示了一组独特的表单控件，用于收集这类支付方式的信息。
- 一个自定义样式的提交按钮。

需要理解这些功能和交互方式在基本体验里应该如何工作，因为不能依靠JavaScript来实现输入限制、验证、反馈或交互。

对于数据输入密集型表单界面而言，我们尤其应该将X光视图主要用于辨认和分离增强界面中的任何下列情况：采用复杂的错误条件反馈，从某一区域到另一区域动态生成数据，根据用户选择显示可选项。在基本体验里处理这些功能需要与服务器交互，所以关键在于找出那些功能上需要服务器提供数据或进行验证的关键工作流节点，并组织好使用体验，以一种符合逻辑的无错方式为用户提供恰当的支持。

从表单的顶部开始，我们能看到一个很直观的配送地址块。它包括一组文本输入框和下拉列

表框，而且在增强体验里不需要使用JavaScript来实现交互。虽然礼品选项（Gift Options）块的设计要用到自定义功能，比如为每个选项显示不同的图标，但数据输入机制却有着类似明确的功能性，就像一个标准的select元素那样，不需要进行繁琐的验证或者引入复杂的业务规则。

在将X光透视法应用到支付版块时，我们面临着一个重要的决策：增强版页面的组织方式是一个由单选按钮构成的标签条，基于用户的交互操作来显示每种支付方式独有的表单控件；服务器根据选择的单选按钮选项动态获取相应的表单控件组信息。但是，在基本体验里，我们并没有动态激活或禁用多个表单字段这样的优越条件。

我们本可以在一个页面里显示一个非常长的表单，将三种支付方式列在它们对应的单选按钮下面，但是这种格式的可用性不高。它将不合理的负担抛给了购物者，让他们去理解技术局限性，填写只和他们选择的支付方式有关的表单字段，并跳过与其他支付类型有关的那些表单元素。这可能会给用户带来错误、困惑、烦恼，最终的必然结果就是放弃结账。

更好的选择是将表单分成多屏显示，将整个流程分成逻辑合理的多步，从而提供服务器端验证以及根据用户输入而定的适当差别体验。这种分离的步骤还有一项好处：它使前面步骤中获取的内容可以被导入后续步骤中，比如，在第一个块里输入的配送地址可以用于生成第三步的支付版块。

带着这个规划，我们下一步的X光流程是观察增强目标设计的每一部分，仔细辨识各种可能性，并将增强组件映射到基本的HTML等价元素上。

前两个版块（配送地址Ship To和礼品选项）里面不存在依赖关系，而且逻辑上是一体的，所以我们会把这两个版块作为基本体验里的第一步，如图2-10所示。

图2-10　增强表单的前两个版块（左图）在基本体验里被组合在一起，成为多屏结账向导里的第一步（右图）

在礼品选项版块的增强体验中，礼品包装（gift-wrap）和礼品卡（gift-card）字段要求是自定义的下拉列表菜单，这样才能为每种选项提供相应的缩略图。通过在基础标记里先添加一个标准的select元素，我们就能为基本体验提供一个可以完美工作（只是缺少缩略图）的菜单版本，然后再在浏览器有能力支持时将其增强成一个自定义下拉列表框。（第17章会详细讨论如何创建自定义的下拉列表框）。

礼品卡附言（gift-card message）只有在购物者选择了一张礼品卡时才有用，因此这个字段在增强体验里会被隐藏起来，直到用户选择礼品卡后再显示。不过，礼品卡附言字段应该包括在基础标记中，这样它在基本体验里才能被访问，如果购物者不选择礼品卡，则会直接跳过这个礼品卡附言字段。虽然显示出一个非必需的表单字段不够理想，但好在只有一个，而且标注明确，所以它的存在不太可能会让用户感到困惑或者给他们增加太大的负担。

现在来处理支付（Payment）版块。在增强体验里，这个界面中的支付类型选项是一排紧凑的单选按钮，并预先在默认页面里填入与最常用的选项（信用卡“Credit Card”）有关的表单字段和内容，如图2-11所示。

图2-11　增强体验里的支付版块默认显示最流行的选项，在这个案例中是信用卡表单

在基本体验里，我们可以将支付选项列在步骤一页面的底部，配送地址和礼品选项的下方，因为从技术上看这一步骤没有事先必须满足的前置条件。但是，从名用户的角度看，支付选项可能会显得格格不入（缺乏逻辑相关性），这可能会造成困扰。另外，我们希望购物者能够从最后的确认页面跳回之前的某一步进行修改，因此如果将配送和支付步骤清楚地分开，修改就会容易得多。（这一要求在增强体验里不是必需的，因为所有内容在单页表单中都可见）。

这样的话，基本结账体验的第二步就会是一张很简单的页面，由三个单选按钮组成各种支付选项。表单在提交后会向服务器发送一个请求，然后根据用户的选择获取正确的支付字段，如图2-12所示。

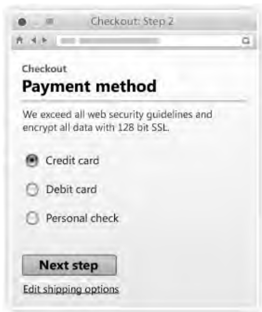


图2-12 在基本体验里用一张简单的支付选择表单获取正确的表单字段

在下一个界面中，服务器将根据用户是选择信用卡、借记卡（debit card）还是个人支票（personal check）选择性输出三张支付表单的其中一张。

如果支付方式是信用卡或借记卡，则会显示一个账单寄送地址（billing address）块，用于验证卡片信息。因为配送地址已经发给了服务器，所以可以给购物者提供一个复选框，如果账单寄送地址和配送地址相同，他们就无需输入这一项。在增强体验里，选中这个复选框后JavaScript会隐藏账单寄送地址字段。在基本体验里，选中这个复选框则会完全跳过下一个界面里要求输入账单寄送地址的表单，如图2-13所示。

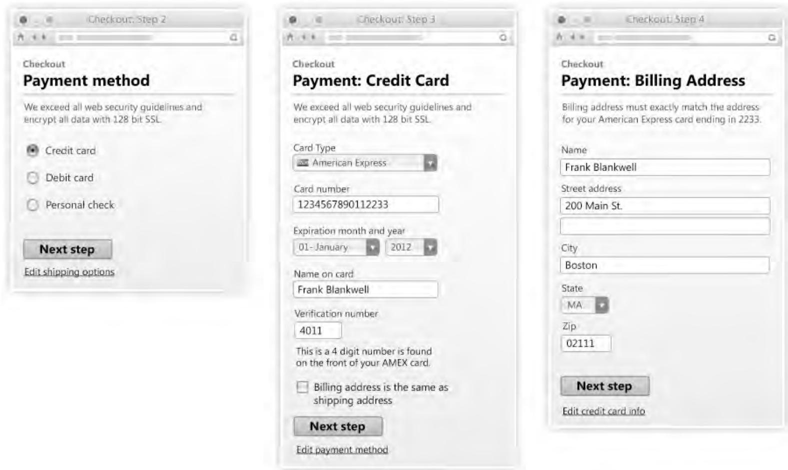


图2-13 基本体验里的支付工作流。那些在信用卡页面（中图）里选中“与账单寄送地址相同”（same billing address）复选框的用户，将会跳过账单寄送地址页面（右图）

基本结账体验包含了一张最终复查页，反馈了在整个工作流里输入的所有数据，其中每一个版块边上都有一个编辑（Edit）按钮，能将用户带回该步骤。因为基本结账体验包含了横跨多张页面的一系列表单，所以用户的选择和输入数据应该在表单提交前展示给他们进行复查，使他们有机会发现和修正错误，如图2-14所示。

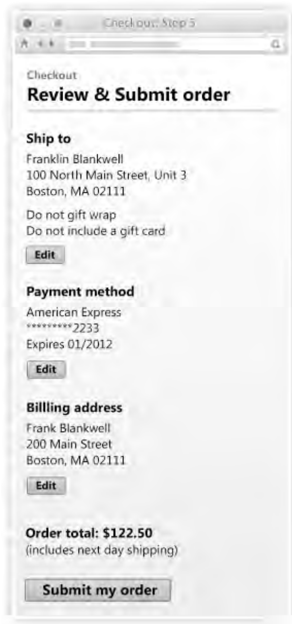


图2-14　基本体验里的订单复查和确认页

这一页只存在于基本体验里。在增强体验中，整个结账流程在一个页面内完成，所以用户自始至终都可以复查和修改信息。

总结一下：基本结账体验包括5个单独的步骤，基于业务规则对用户工作流进行分流，并一路验证，如图2-15所示。

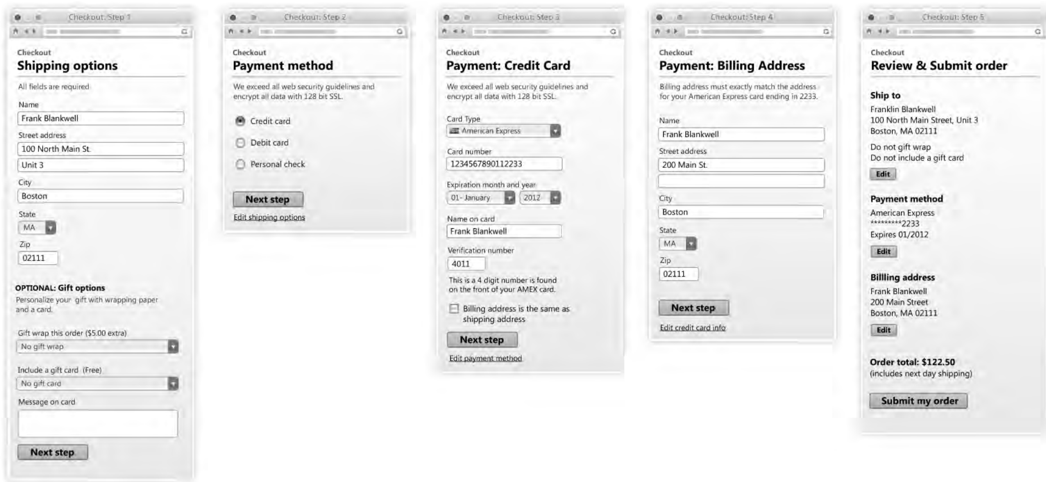


图2-15　基本结账体验的5个连续步骤

(1) 配送地址和礼品选项。
(2) 支付方式（信用卡、借记卡、支票）。
(3) 支付细节（根据步骤(2)）中用户的选择显示自定义表单字段。
(4) 支付账单寄送地址（只适用于信用卡和借记卡这两个选项）。
(5) 复查订单。

2.3.2　标记表单以确保可访问性

要以最具可访问性的方式构建结账表单，关键在于选择正确的HTML标记结构，并且要特别注意考虑潜在的语义。我们是否已经尽最大可能为表单元素的组织和编码增加描述性和可用性？比如，每个表单元素都应该附带一个恰当的标签，并尽可能提供最佳的用户体验（表现在易于使用以及进行恰当的限制避免错误上）。

在基础标记中，每一张结账页开头的form标签都必须将action属性指向将会处理此表单的服务器资源（比如一个PHP文件）。在结账过程的每一步，表单控件要根据逻辑关系分组到一个fieldset元素中，每个input元素上方的标签只能用最少的样式（不要设置表单元素的浮动或者宽/高属性）。在每一页的底部要提供一个用来提交表单的下一步（Next Step）按钮，将用户引至结账过程的下一个步骤，还有一个返回（Back）链接，用于将购物者带回之前的页面。

我们的增强结账表单设计包含了一些内嵌缩略图的自定义样式下拉列表菜单：礼品包装选项、礼品卡选项和一个自定义信用卡选择工具。这里的关键是要在基础标记里使用一个基本的HTML select元素，这样才能确保任何用户都能选择相关的所有功能，然后在某种浏览器通过能力测试后再用自定义样式增强那个select元素，如图2-16所示。



图2-16　基本体验使用了一个标准的HTML select元素（左图），在增强体验里则转变成了一个带有缩略图的自定义样式select（右图）

注意　要详细了解如何制作自定义样式的下拉列表框，参见第17章。

这个自定义样式的“提交我的订单”（Submit My Order）按钮是体现正确选择标记重要性的另一个例子。最新的HTML规范（4.01）提供了两种元素：一种是将type属性设置为submit的input元素，另一种是button元素。两者天生就是用来提交表单数据的，但是button元素在XHTML一个名为XHTML-MP（mobile profile，移动配置）的子规范中被略去了，因此如果浏览器的渲染依照的是XHTML-MP规范，就不会支持这个元素。相反，input元素则能被任何一种浏览器支持（只要它遵循HTML 2或之后的规范），能在包括小众和旧式移动浏览器在内的最广泛的浏览器中工作。我们想要确保任何交易过程都能可靠地在所有地方工作，服务于任何人，因此推荐在这里使用input元素，如图2-17所示。



图2-17　基础标记里使用了广泛兼容的input元素（左图），在增强体验里则有着自定义的样式（右图）

通过应用一些自定义样式让input提交元素的外观和行为像一个按钮，任何地方的用户都能得到满意、可预测及可靠的体验。（第14章将描述开发自定义按钮的推荐流程。）

2.3.3　添加限制与验证

确保购物者在结账过程中输入有效信息对于完成交易来说非常关键。作为设计者，可以使用3种技巧来确保输入的是有效数据：提供操作说明和帮助文字来展示正确的格式，添加限制来避免无效输入，以用户反馈的形式实施验证逻辑（在屏幕上告知用户他们的输入无效，需要进行修正）。

提供操作说明和示例数据是我们这些验证选项中最友好的一种。从一开始就提示正确的格式对于减少无效数据而言大有帮助。我们建议或者直接在页面上显示帮助内容，或者根据上下文的具体情况在特定字段边上跳出辅助提示，以此保持整个体验的清爽，并帮助用户把注意力集中在手头的问题上，如图2-18所示。



图2-18　操作说明能帮助减少数据错误。在基本体验里他们应当被放置在页面上（左图），在增强体验里则可以选择性地显示（右图）

还可以限制用户必须输入有效的数据值，方法是提供那些只接受有效输入的控件和增强组件（比如复选框、单选按钮、下拉列表框、自动完成式文本输入框、滑块或者日历式日期选择器），并运用渐进式展开（progressive disclosure）的方式，只显示手头任务所需的表单元素。

如果设计需要用到文本input这样的自由文本输入控件来输入信用卡号码，我们可以在基本体验里提供操作说明文字，然后配置服务器端的验证来检查输入值，确保它是正确的类型（例如数值型），在可接受的输入值范围内（例如15位），如有必要还可以增加更为复杂的算法辨识出数字模式以匹配特定的卡片种类，如图2-19所示。



图2-19　基本体验里的信用卡号码验证需要提交表单（左图）。在增强体验里，输入的内容可以用JavaScript即时验证，错误消息以弹出提示的形式显示（右图）

基本体验需要与服务器交互才能提供验证反馈和根据相关上下文环境隐藏/显示表单字段。用户点击“提交我的订单”按钮后，服务器可以验证表单数据，检查是否有必填字段漏填或者字段包含无效数据值，然后让用户返回到相关页面，并提供明确的消息，告知哪些地方需要更正。服务器还可以协调接下来显示哪些表单字段，比如根据用户选择的支付方式该显示哪张支付表单。

在增强体验里，大多数这些用户限制和验证行为可以使用Ajax实时完成，反馈则在用户打字或选择完成时提供。这个技巧让我们能够在基本和增强体验里重复使用同一套服务器端验证规则。

2.3.4 组合基本和增强体验

这种单页的增强体验在基本体验里被分解成那么多步，看起来似乎很难在两种体验的基础标记里使用相同的代码。不过，我们可以向每个人输出基本体验里的第一步，运行能力测试，然后借助JavaScript（更确切地说是Ajax）和CSS的力量将5个基本步骤的基础标记组合成单一的增强页面。另外，为了加快网页载入，服务器还可以检查能力测试所设置的cookie，看看这个浏览器是否已经通过了测试，然后再为增强体验的用户组合整个页面。

还可以利用已创建的后端逻辑，同时为基本和增强体验输出合适的表单并执行验证规则。在这种情况下，服务器逻辑会被配置为只读取请求数据，增强体验则会采用Ajax生成请求，而基本体验会使用标准的网页请求。

2.4 案例 3：预算计算器里的交互数据可视化

我们的下一个研究案例可能是视觉性最为丰富的一个例子：一款用于规划财务预算的动态预算计算工具。它使用一组交互滑块进行即时数据计算，并以显示带有悬浮提示的交互性饼图（pie chart）和月度及年度总额的形式提供实时反馈，如图2-20所示。



图2-20 预算计算器的目标增强设计

这个工具第一眼看上去可能不会立即让人感觉适合使用渐进增强方法。但是通过X光透视可以发现，这个工具的核心功能非常直观，并且适合使用通用设计方式：那些高级滑块元素只是让

用户在若干类别里输入数值，而一览表和饼图显示了这些数值计算后的总和，并按照类别和总金额显示它们的相对权重。

2.4.1　选择预算线组件的基本标记

这款预算计算器使用一个滑块组件来设置和调整每个预算线[①]（budget line）项目的数值。对于一个增强滑块组件而言，可选择的“最佳”基础代码有不少，哪种元素是正确的选择要根据现有选项的数量和长度而定。

- 对于有2~5个简单选项（评分：很好、好、一般、差、不适用）或10以内的数值等级来说，单选按钮组是一个不错的选择，因为它能显示所有的选项以方便浏览，而且还内置限制性，避免用户错误输入。
- 如果连续体范围更大或者更个性化，致使滑块所在的范围有5~20+个固定选项（例如金融评级里的级别：AAA、AA+、AA、AA–、A+、A、A–、BBB+、BBB等），下拉列表框则是最佳选择，因为它提供了良好的限制性，数量更大的选项列表还可以使用optgroup元素进行视觉分组。至于缺点，下拉列表菜单的选项不容易浏览，因为默认情况下它们被隐藏于视野之外（包含在一个下拉菜单里），而且可能需要滚动查看。
- 当用户可以输入各种数值时（最高价格$0~1000），文本输入框就很合适了，但是要注意它没有原生方法可以对数值输入进行限制，因此这种方式需要在服务器端进行验证。

我们会为每一个预算滑块使用一个简单的文本input，因为希望用户能够输入各种美元数值，包括使用小数点来提高精确度。

这个预算计算器的基础标记以一个form标签开头，action属性指向处理该表单的某个资源。每个预算类别线项目都放在一个div里，其中包含一个label，一个用于输入美元预算值的关联文本input，以及一个显示该类别百分比计算值的span。我们给每个类别编写的标记看起来就像这样：

```
<div>
  <label for="category-1">rent/mortgage</label>
  <input type="number" min="0" max="5,000" value="1,500" id="category-1" />
  <span class="budget-percentage" title="Percent of total budget">48%</span>
</div>
```

这个输入框有一个number属性，还有额外的min和max属性，以设置它能接受的高低范围。一开始它的实际数值和百分比计算值都会是空白，之后随着每次页面刷新而生成必要的值。

> **注意**　这个输入框的number类型是HTML5规范里的一个新属性，不过现在可以安全使用，因为不支持它的浏览器会默认将type属性视为text。第3章会更详细地讨论输入框属性。

① 在经济学里，预算线是指给定商品价格和收入等级的情况下，一名消费者所能负担的任意两种商品组合所构成的一条线。——译者注

在基本体验里，表单需要在类别列表底部放置一个“计算预算”（Calculate Budget）按钮来提交表单数据，并在服务器上计算总金额及类别的百分比，如图2-21所示。

Quick Budget

Sign up | Log in

rent/mortgage 1,500 48%

car/transportation 300 10%

utilities 75 2%

groceries 300 10%

entertainment 375 12%

spending money 150 5%

retirement savings 400 13%

$36,000 yearly

$3,000 monthly

Calculate budget

图2-21　基本版计算器提供了简单的文本输入框、百分比计算值和总金额，以及一个用来在服务器端更新数据变动的按钮

我们不得不适应一种错误情况：用来输入预算数额的文本输入框并不能原生提供很好的限制来防止人们输入字母或特殊符号。基于这个原因，每次提交表单时服务器都需要对输入进行验证。

我们还可以通过在基本体验里提供一组“空白”文本输入框，实现添加新预算类别这个功能。就像用“提交”（Submit）按钮重新计算预算那样，“添加”（Add）按钮会提交整张表单，然后使用新的类别和计算值重新加载页面，如图2-22所示。

Add a category

Name

Amount

Add

图2-22　这张小表单在基本体验里添加一个新类别并重新加载页面

通过使用这两种简单的表单来收集预算信息与添加新的类别，我们就能够支持所有的关键功能，提供令人满意和内容丰富的基本体验。

2.4.2　从基础标记开始创建可访问的滑块

虽然基本体验里的简单表单字段能够完成任务，但是对那些使用有能力浏览器的用户来说，增强体验里具备触摸质感的滑块控件能鼓励他们进行更多探索，而且非常有乐趣，如图2-23所示。



图2-23　用颜色编码的交互滑块在拖动时会更新美元数字反馈和饼图的视觉形象

在渐进式增强和X光透视的背景下，要让基本和增强体验协同工作需要遵守一些关键原则。

首先，要制作一个收集用户输入或数据输入的增强组件时，我们都会使用一种“代理”模式来制作它。举个例子，在页面加载时，用来创建滑块的增强脚本会查找对应的基础input里是否已经存在数值，如果存在则使用此数值来设置滑块操纵杆的初始位置。滑块移动时，脚本会重新设置input的值予以匹配，而且这个脚本会不断地保持两者同步。在许多情况下，我们会让滑块和可编辑的input同时出现在页面中，让用户自己选择用哪一个。这种方式的好处是服务器能够以相同方式处理基本和增强体验里的数据，因为它只和input交互。

其次，因为没有HTML元素能够原生映射出滑块的语义，所以我们会将非语义的标记用于这个增强组件，这就意味着我们需要加入可访问性功能来模拟用户期待的那些原生行为。在这种情况下，我们会在代码里用一连串的div来制作滑块容器、轨道和操纵杆。但是，使用辅助设备的用户不会知道这堆div是一个交互性组件，除非我们加入可访问性功能将它“翻译”出来。具体地说，会添加一个值为slider的ARIA role属性标明这是一个滑块组件，还要用其他一些属性设置滑块的最小值、最大值和当前值，这样新版的屏幕阅读器就能解读这个组件。对那些使用不支持ARIA功能的旧式屏幕阅读器的用户而言，能够与原始的输入框进行交互则成了必需的可访问性功能。

最后，因为滑块依靠拖动行为来移动操纵杆，所以还必须要考虑许多新式的移动设备（比如iPhone、Palm Pre和Google Android）用拖动事件实现一般的屏幕各向滚动。为了保证滑块在那些设备上仍然可用，滑块脚本必须能让用户通过点击操纵杆获取焦点，然后在轨道的其他位置再次点击来设置新值。类似的行为还支持键盘访问：Tab键让滑块操纵杆获得焦点，方向键移动滑块到一个具体值。

2.4.3　制作饼图

能够看到饼图随着预算滑块的移动即时更新，是增强体验里的一项特色功能，而且确实增强了交互性。但是依靠Adobe Flash等插件或某个服务器端的工具来生成一张静态饼图并不包括在我

们为之奋斗的包容性渐进增强原则之内。幸好，我们可以采取一种更加通用（而且向前兼容）的方式，即使用HTML5的canvas标签在众多现代浏览器里生成原生的HTML图表，如图2-24所示。（甚至连Internet Explorer都可以不借助插件渲染出这张图表，只需使用一个谷歌的脚本，名为ExplorerCanvas：http://excanvas.sourceforge.net）。

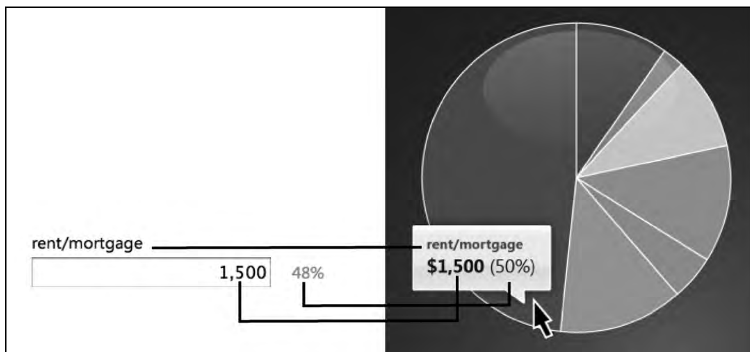


图2-24　在增强体验里生成饼图提示使用的值和基本体验里使用的值相同：类别标签、数值和百分比

饼图中的值直接取自生成每个输入框边上百分比反馈信息的JavaScript代码，然后传给另一个脚本以生成canvas图表。（第12章将详细介绍用canvas制作可访问图表。）

用户将他们的鼠标停在饼图某个部分的上方时，饼图的工具提示会显示这部分的标签、预算美元金额和百分比。这些数据只是呈现“增强”表格（以及基础标记）中标签、输入和百分比值的另一种方式。

注意　第10章将详细讨论如何创建可访问的自定义提示。

从可访问性的角度看，一张饼图无论如何渲染都无法为盲人用户提供有用的信息。因此，我们会在增强体验里为屏幕阅读器用户添加一点背景信息，让他们明白canvas标签及任何与图表有关的标记都纯粹是展示性的。我们会将canvas放置在一个div容器内，并添加一个名为img的role来表示canvas扮演着一张图片的角色，另外还会在aria-label属性里提供操作说明性质的消息文字，解释这张图表的用途和它的数据值。对于那些使用不支持ARIA的旧式屏幕阅读器的用户，我们会在一个span标签里提供第二则消息（此标签会对非盲人用户和支持ARIA的屏幕阅读器隐藏），解释这张图表的标记纯粹是展示性的，并附上一个“跳过”链接，这样他们就可以轻松越过那个标记。

在基本体验里，百分比反馈所表现的信息和文字版的饼图没有区别，所以你可能完全不需要将饼图包括进来。不过，如果对图表有需求，可以在每次重算预算时生成一张静态的饼图图片，然后用一个image标签添加到页面中。

现在就有了一个完全可用且可访问的预算规划工具，可以在任何设备上工作。我们的工具箱里则多了一些有趣的X光技巧，可以在其他环境里再次使用。

2.5　案例 4：支持功能完备浏览器应用程序的各种功能——照片管理器

我们最后的案例是一个照片管理器界面。作为Web 2.0时代的一个新近产物，它所具备的复杂应用程序功能之前只有在桌面端才能见到，现在则复制到了Web浏览器中，如图2-25所示。



图2-25　基于Web的照片管理器应用程序

就像这个例子所展示的，今天的许多程序和在线工具都包含了类似的Ajax驱动型复杂应用程序功能，包括用于导航和显示内容的可伸缩/缩放面板，简单的单个物体拖放手势，通过鼠标拖动选择或者Control/Shift键进行复杂多重选择的能力，以及用Ajax实现的即时编辑操作。

这个设计体现出了如此多样的交互性，你几乎都难以想象如何能在基本体验里支持所有这些功能！事实上，在用X光透视如此复杂的界面时，很大一部分工作是仔细审视其中的功能并确定该如何处理它：选择功能也许有一个简单的等价元素；复杂一些的交互和手势可以被组织成多步骤或者分阶段的工作流；另外，在一些情况下，我们还可能会认定某些类型的高级功能过于复杂，无法保证所有用户和设备都能看到这些功能的等价基本HTML。

就像处理其他目标设计一样，我们首先会站在高处观察这个照片管理器，理解内容和功能的分组情况，寻找是否存在复杂的数据提交先后顺序，确定对所有用户来说哪些内容和组件是实现网站功能所必需的，然后将它们映射到等价的基本HTML以提供支持。

2.5.1　制作全局导航元素的标记

我们会从整体的页面布局着手，因为对它进行X光分析足够容易：顶部包含产品名称和主导航版块（“组织”Organize，“上传”Upload，“购买”Shop，“账户”Account）的长条可以被编码为一组标准链接。在基本和增强体验里，每个链接都会重新载入整个应用程序窗口，因为该应用程序的相应版块会包含不同的功能。我们会在基础标记里将一个标题元素（`h1`）用于产品名称“Filament照片管理器”（Filament Photo Manager），然后在增强体验里给它添加样式，使它看上去更精美，如图2-26所示。



图2-26　在基本体验里看到的标题和全局导航

至于随版块而定的二级导航，左侧的导航面板也应该在基础标记里被编码成一组标准链接，用标题来区分和归组两种类型的内容（照片和专辑），如图2-27所示。

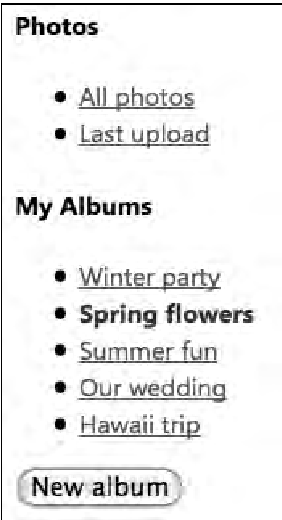


图2-27　左侧的导航面板被编码成一系列标题和链接列表

在增强体验里，用户点击左侧导航面板里的某个链接时，我们想以一种类似应用程序的无缝交互手段在右侧面板里生成相应的照片。我们会用JavaScript来截获链接的`href`值，然后构建一个Ajax请求，从所选专辑里取回照片放入右手边的面板里，如图2-28所示。

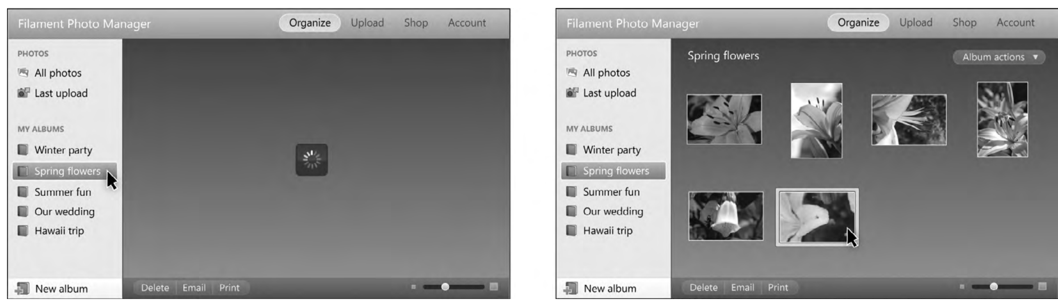


图2-28　在增强体验里，点击左侧面板里的某个专辑名后，Ajax会在右侧面板里载入网格式的专辑照片

在左侧导航面板的底部还有一个“新专辑”（New Album）按钮。在增强体验里，点击这个按钮会在左侧面板里动态创建一个新的专辑链接，名字是默认的，可以进行编辑；在基本体验里，这个按钮将用户导航至另一张页面，里面有一张用于命名专辑的简单表单，如图2-29所示。



图2-29　增强版的“新专辑”按钮在左侧导航面板里创建出一个可编辑的节点；在基本体验里，这个按钮导航至另一张表单页来命名专辑

2.5.2　支持专辑和多张照片的复杂交互

在增强体验里，每张专辑都显示为一个干净、简单的界面，顶部是专辑名，次级操作则被放入右上角的菜单里。底部的工具栏里有一组操作选中照片的按钮，以及一个用来改变缩略图大小的滑块，如图2-30所示。



图2-30　增强体验里的专辑信息版块在屏幕顶部提供了“专辑操作”（Album Actions）菜单，底部是一个操作选中照片的工具栏（“删除”Delete、“发送邮件”Email、“打印”Print）

有一个功能让增强版的照片管理器真正感觉像是桌面应用程序：在专辑网格视图中，你可以通过在一组照片周围拖动套索（lasso），或者按住Control或Shift键并点击一组照片来选择多张照片。选择完成后，这个界面允许用户点击底部的操作工具栏来编辑、删除、通过邮件发送或者打印所选照片，还可以将照片拖放至左侧面板里的某个专辑名上，把它们添加到该专辑，如图2-31所示。



图2-31　可以选择多张照片，通过拖放将它们添加到某个专辑中，就像一个桌面应用程序那样

这些富交互手段乍一看可能与渐进增强的方式不相容，但是事实上你可以很容易地使用标准HTML表单元素来提供同样的功能。

基本体验里的专辑视图看上去样式要少很多，却同样提供了所有的功能。专辑页以一个标题开头，接下来是一排专辑操作按钮，用来将用户送至其他多个页面以完成或确认这些操作，如图2-32所示。



图2-32　导航进入基本体验里的专辑信息页后，源代码顺序中首先显示的是专辑名称和"专辑操作"标题

在考虑鼠标拖动和Control键点击这些复杂的交互方式时，X光透视显示出它们的作用仅仅是让用户选择多张照片，我们可以很容易地用原生表单元素在基本体验里复制这个。在基础标记里，每张照片的代码都可以附上一个包含图片名称的`label`，以及一个用于选择这张照片的`checkbox`。专辑的标记放置在一个`form`标签的内部，用多个`submit`按钮执行删除、通过邮件发送和打印操作，另外还有一个下拉菜单与"添加"（Add）按钮的组合，用来将选中的照片指定给某个专辑，如图2-33所示。

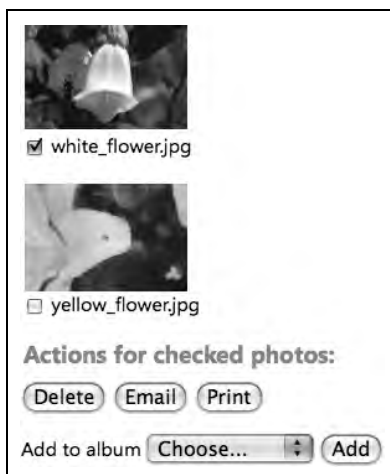


图2-33　在基本体验里，复选框方便用户选择多张照片，以及用一排按钮对该照片组执行操作

这种有着表单样式的交互手段在基本体验里是完美可用的，甚至还可能会得到那些不太习惯使用拖放手势的用户们的偏爱，更不用说屏幕阅读器用户，或者因为手机是多点触摸界面而根本无法执行拖动操作的用户了。（我们总是喜欢在能力测试里提供一个备用链接，让用户从增强体

验回到基本体验。这个经典的例子说明了即使用户使用的是有能力的浏览器，他们也可能想要简化体验，提供一个链接能让他们选择最符合自身需求的版本。）

用户在增强体验的专辑视图里双击一张缩略图后，专辑缩略视图会切换成全尺寸的照片细节图，如图2-34所示。



图2-34　在增强体验里双击一张照片的缩略图后，Ajax会在右侧面板里载入一张全尺寸的照片

专辑网格和照片细节视图之间的这种平滑渐变只有通过JavaScript才能实现。为了让基本体验里的用户能够访问照片细节以及相关的编辑工具，我们会在每一张缩略照片上放置一个链接，它会打开一张单独的HTML照片细节页。和上面的专辑一样，我们会用Ajax取回和照片细节页的基础HTML标记完全一样的代码，填入增强版的面板中，加以适当的样式处理，所以说两种体验里的标记没有区别，如图2-35所示。



图2-35　增强体验里的照片细节视图

增强体验中的照片细节页里有照片的名称，一个“返回专辑”（Back To Album）按钮，沿着这个按钮下去有一个操作工具栏，可以方便地旋转、裁切、删除、通过邮件发送和打印这张照片。在基础标记里我们会创建一些标准表单提交按钮，让用户能够对照片执行这些操作，而同样的按钮在增强体验里仅仅是加了样式以及位置不同。增加快捷键（比如将删除键映射到删除照片上）和鼠标手势（比如支持用拖放将照片添加到左侧导航列表的专辑中）会提升交互的丰富性，模拟出原生桌面应用程序才有的行为，如图2-36所示。

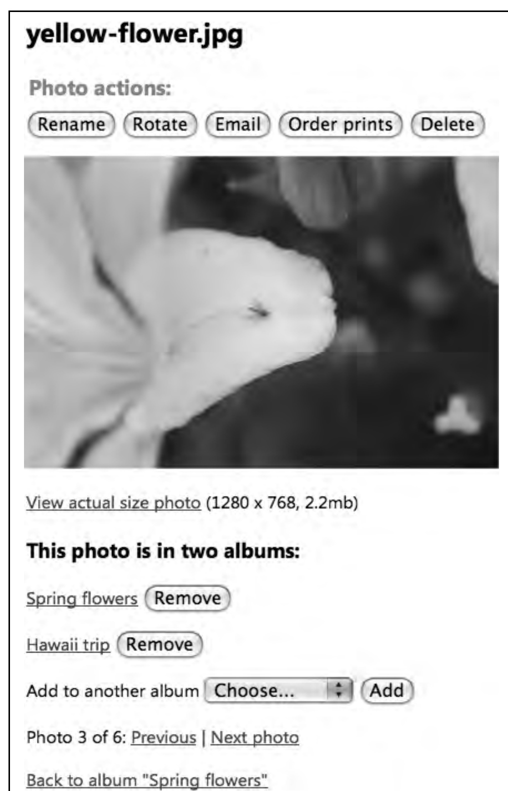


图2-36　在基本体验里，点击某张带链接的照片会将用户导航到一张单独的照片细节页，内含优化过的工具和导航项

在基本体验里，我们会在照片下方添加几个独特的功能来增强可用性。首先，提供照片所在专辑的清晰反馈信息，并在每张专辑的旁边加上一个“移除”（Remove）按钮。此处还提供了与专辑网格中相同的“添加到专辑”功能，以便将照片加入到其他的专辑中。这里的每种操作都会提交一张表单并重新加载页面以提供准确的反馈信息。

屏幕最下方的导航链接可以用于浏览专辑里的后一张或者前一张照片，以及返回专辑网格视图。将基本体验分解成多张页面时（就像现在这样），提供额外的导航选项来改进用户的整体流程总是没错的。

“裁切”（Crop）按钮到哪去了

你可能会注意到我们在基本体验里省略了“裁切”按钮。这是有意而为的。有时在X光透视时你会碰到一些功能，这些功能用标准HTML表达时过于复杂，难以用于基本体验。对我们来说，裁切工具就是这样一个例子：我们如何才能在基本体验里重现出裁切框的拖动缩放（drag-and-resize）功能？要是认定这个功能绝对是必需的，可以探索一些访问解决方案：比如，用服务器计算图像的总像素尺寸，或者向用户提供一组文本输入框，让他们输入图像左上角和右下角的像素坐标。但是相比简单地建议用户在基本体验里先适当裁切图像再上传，上面无论哪一种选择的用户体验似乎都要糟糕得多。

我们相信在任何网站或应用程序的基本和增强版本之间，努力实现尽可能多的平等是最好的方式，但是也承认当可用性或开发努力毫无意义时，某些功能需要从基本体验里砍掉。

2.5.3　创建自定义表单和叠加

除了专辑之间及专辑网格与细节视图之间的导航外，添加一些小的叠加（overlay）和对话框在许多情形下会让增强体验真正显得具有响应性，像个应用程序。比如，点击操作工具栏里的“发送邮件”按钮可以在照片上方滑动显示出半透明的叠加层，提示用户输入收件人和邮件内容，再用Ajax发送出去，如图2-37所示。



图2-37　在增强体验里，“邮寄照片”（Email Photo）表单显示在一个叠加层中，由Ajax附加到页面上

要在基本体验里实现这个功能，“发送邮件”按钮应将用户导航到包含邮件表单的一张单独页面里。（基本和增强体验可以使用相同的表单基础标记，在增强体验里先由Ajax获取，然后修

改样式，使其看起来像叠加设计。）在邮件成功发送之后，网页应该刷新，并显示出一条确认消息和一个返回照片细节页的按钮或链接，如图2-38所示。

图2-38　在基本体验里，“邮寄照片”表单是一张单独的HTML网页

这个为核心屏幕类型和功能操作提供导航和功能操作支持的方案遵循一组通用模式：在基本体验里将工作流分为多屏，在增强体验里则重复利用基础代码，并辅以叠加和原地处理功能（in-place feature）。

2.5.4　创建返回按钮支持

我们觉得，在创建复杂的网站或网络应用程序，特别是用到Ajax时，还有最后一个关键要点值得仔细考虑：浏览器的返回按钮将会如何工作。

虽然网络用户对浏览器里类似应用程序的复杂交互方式正变得越来越习以为常，但是他们仍然期待某些浏览器的原生传统能继续保持，包括用浏览器的返回按钮能导航至前一页这个概念。当一款应用程序使用Ajax时，返回按钮在默认情况下不能识别Ajax生成的临时步骤，反而会将用户导航回它最后加载的真正“网页”中（很多时候那会是登录页面），这可能会让用户十分困扰和恼火。即使是在那些没有使用Ajax编程的网站或应用程序里，用户有时也可能会将页面中一些大块的可切换内容面板看做一个新“页面”，他们会点击返回按钮，期望能“返回”到那里。

在照片应用程序案例中，我们预计人们很可能期望返回按钮能将他们带回到之前的导航视图中（专辑1→照片细节A→照片细节B→专辑5），但不会撤销他们在这个过程中执行的任何操作（添加到专辑、发送邮件等）。简单地说，通过添加一个独一无二的URL#号串（URL hash，URL中以#开头的一串字符）并使用JavaScript在每次用户导航到某个独一无二的专辑或照片视图时进行检查，返回按钮就能像预期的那样工作，而且所有的常规浏览器操作（比如收藏和用邮件发送某个链接）同样也能像预期的那样工作。

每个网站都不同，因此具体该如何让返回按钮的行为可预测且有用，要根据相应的系统模型而定，但它在让应用程序良好工作方面扮演着重要的角色，需要得到重视。第9章介绍如何制作标签页组件时，将讨论了用这种方式进行状态追踪的具体技巧。

2.6　在实践中运用 X 光的核对清单

当我们将功能映射成标准HTML，创建完基础标记，添加上增强信息，并反复多次进行测试和完善之后，X光方法的最终步骤是过一遍下面这份问题核对清单以检验进展。

- 基本体验是否只靠HTML就可用而且功能完整？
- 基础标记是否编码了增强体验所需的一切信息，包括布局和结构？
- 在基本和增强体验里，页面的阅读和工作顺序是否合理，是否突出了最重要的内容和功能？
- 如果只使用键盘命令是否可以在页面中导航？表单元素是否可用（可以执行选择和提交操作）？
- 网页在使用屏幕阅读器访问时是否能看明白？用户能不能单独在页头结构里导航？
- 无论是因为个人偏好还是增强版本由于某种未曾预料到的原因出现了故障，是否有一种明确的方式能让所有用户（包括使用移动设备和屏幕阅读器的那些用户）在基本和增强体验中相互切换（比如一个“浏览低/高带宽版本”的链接）？

虽然这些问题可能无法揭示出为每一种浏览器体验提供可访问性和渲染支持时会出现的所有异常情况，但它们突出了许多关键要素，会帮助我们实现普遍访问这一目标。

我们只介绍了在所有可能的网站和网络应用程序设计中能遇上的全部情况的冰山一角。但我们希望这一概述能够让读者了解到，有多少增强交互方式能被转化成简单的HTML，用于基本体验。更重要的是，希望它能在你思考如何用渐进增强方法处理复杂的应用程序时，带给你灵感。

渐进增强（总体而言）和我们的X光方法及能力测试（具体而言），只有在你严格采用最佳编程实践方式编写HTML标记、CSS和JavaScript时才能有效工作。接下来的三章会介绍一些关键的要点和功能，我们相信理解它们对于正确实施渐进增强设计至关重要。

在此基础上，我们会接着介绍能力测试框架工作原理的具体情况，以及如何完善甚至扩展它，从而以更有创意的方式满足你自己的特殊需要。

之后，深入一些组件的内部细节来演示真正的实战技巧，借助这些技巧不仅可以让渐进增强方法适用于今天的浏览器，还能用于将来的。

第 3 章

编写有意义的标记

现如今，很多诱惑会让你忽视编写恰当的标记和结构：网页技术军火库里的许多武器能彻底改进HTML，有经验的设计师和开发者能在一堆div和span上应用CSS和JavaScript来实现层级，美化元素或者通过脚本使这些元素按照他们的意愿工作。但是，拿掉CSS和JavaScript会发生什么？内容层级是否依然存在？最重要的信息是否出现在页面的开头部分？导航在哪里？寻找起来方便吗？所有功能是否仍然能正常工作？

本书建议将渐进增强和能力测试整合进开发过程，以此确保一切内容和功能都可以覆盖到所有使用可上网设备的人。但是，要运用这种方法，网站必须建立在一个恰当的基础之上，那就是干净、组织良好而且架构合理的语义标记。

什么是语义标记？在2001年《科学美国人》的一篇文章“The Semantic Web”（语义万维网）中，Tim Berners-Lee[①]将其定义成一种“给网页中有意义内容创建结构”的方式（Tim Berners-Lee、James Hendler和Ora Lassila，“The Semantic Web”，《科学美国人》杂志，2001年5月，http://t.cn/apHkZC）。换句话说，语义标记就像是词汇表、语法、发音和行文方式，帮助浏览器（以及延伸到的用户）理解网页内容。我们发现，通过将语义化HTML标签以聪明的方式（这些方式甚至能让缺少CSS和JavaScript的基本体验都一目了然并且可用）组合在一起，我们就能够表达出惊人复杂和强有力的想法。

编写可以在所有浏览器上工作并且能正确支持增强的基础标记，需要仔细考虑该使用哪些元素，以及将它们用在何处才最有效。标记就像任何一种语言一样，如果没有扎实的语言结构和规则知识，它就只是一堆声音和符号的集合而已。

这一章会分析最常用的HTML元素，并描述明智的语义选择能够怎样改善浏览器的解读，并优化任何可上网设备上的用户体验。我们会分4个部分考虑语义标记中的选择。

- **标记文本和图像**：它们是任何网站的基石，只有尽可能地标记准确才能发挥最佳作用。这个部分探索了一系列语义标签，从而智能分辨网页内容及赋予层级意义。
- **标记交互元素**：表单和其他交互元素包含了非常特殊的功能，选择正确的标记对于支持预期行为和促进最佳用户交互而言至关重要。
- **在网页中创建上下文关系**：归组内容项目和相关表单对象的方式是一个良机，能让你帮

① 蒂姆·伯纳斯-李（Tim Berners-Lee）被公认是万维网的发明者。——译者注

助浏览器（和受众）理解你传达的讯息和目的。

- **建立一份HTML文档**：最后，使用标记来定义HTML文档和标明网页自身也是一套方法，可以对内容的意义进行编码，并将解读方法告知浏览器或其他用户代理。

随着内容的深入，我们会介绍如何解读当代设计难题并将它们分解成独立的部分，这样就可以建立最佳的内容框架和标记，然后在这个基础层上添加CSS和JavaScript增强信息。

3.1　标记文本和图像

作为设计师，我们知道每一种美好的体验都始于清晰、定义良好的内容。当你完成了网站描述或者应用程序的工作流之后，把内容翻译成标记只不过是搞清楚要用哪些元素而已。

一些元素（比如标题）有双重用途：各种浏览器渲染它们的方式相当一致，使得它们在视觉上与其他内容区别明显，在底层它们则向辅助技术（比如屏幕阅读器）和机器（比如搜索引擎）提供操作指南，使其无需借助设计里的任何视觉线索就能解读内容。

接下来的部分介绍了我们推荐的那些语义元素，通过它们能将有用的意义赋予基础文本、图像和富媒体。

3.1.1　用于标记有意义文本的元素

用来呈现印刷文字的惯例（包括段落、标题和引用文字，还有很多）已经存在好几个世纪了，它们能帮助读者解析和理解某个文本的层级和意义。这些规则在网络上与在古腾堡发明印刷媒介时同样成立，语义标记则提供了一系列丰富准确的标签来帮助将意义赋予文本网页内容。详尽准确地运用这些元素，能够显著改善浏览器或搜索引擎解读并呈现文本内容的方式。

1. 标题

标题元素的作用是标记标题和子标题，将内容分割成多个部分并在页面中创建层级。

标准的语义标题元素从h1开始，它代表了页面里的最高层级。后续每一级标题的重要程度依此递减，直至h6，它是最小的可用标题。

```
<h1>最重要的标题</h1>
<h2>次重要的标题</h2>
<h3>次级标题</h3>
<h4>次级标题</h4>
<h5>次级标题</h5>
<h6>最小的次级标题</h6>
```

对搜索引擎和屏幕阅读器而言，标题还在网页意义和可访问性方面扮演了一个重要角色。

虽然搜索引擎不公开它们的算法，但是许多人相信一些搜索引擎基于内容层级里的标题来解读内容的相对重要性，也就是说相对于标准的连续文字，标题元素中的文字会被赋予相对更高的排名权重。相比之下，一个段落、div或者span即使将样式修改得看起来像个标题，也不会被赋予同样级别的重要性。

视力残障人士使用像JAWS、NVDA、苹果VoiceOver和WindowEyes等屏幕阅读器来访问网页

内容和功能。举个例子，许多屏幕阅读器利用标题元素快速浏览内容和导航。它们让用户能够简单地通过击键来遍历某张网页中的标题元素，其中一些还提供了列出这些标题的专用面板，用来进行快速目录导航（www.w3.org/TR/WCAG-TECHS/H69.html）。

另外，因为标题能帮助进行页面导航和视觉性内容解析，所以良好的做法是用逻辑一致和不跨级的方式使用它们。比如，在h1的后面使用h2，而不是h3。虽然省略了标题层级的标记也能通过验证，但它可能会给那些试图理解内容组织方式的用户带来困惑。

为了确保你的标记有效地使用了标题，不妨考虑在禁用CSS之后查看网页。内容层级（包括任何交互功能）应该是清晰的，就像大纲一样。如果没有明确且可预测的标题结构，用户浏览网页时就可能很难找到重点信息和内容结构。

在增强体验里，标题可以用CSS和JavaScript转变成交互组件上的可点击标签，比如手风琴式（accordion）或分组式标签页组件。这些增强功能同样依赖于恰当且一致的结构。比如，在标签页栏里与其他标题平起平坐的标题应当使用与前者等级一致的h元素。（后续章节会详细讨论这些转变标题的方式：手风琴式展开参见第8章，树形控件参见第11章，标签页参见第9章。）

2. 段落文字

段落是块级元素，用于在网页中把叙述性内容分解成多个想法或观点。

```
<p>这是一段文字，包含在一个段落标签中。</p>
<p>我是第二个段落。我用两句话表达一个完整的观点。</p>
```

我们不建议用段落元素标记任何不属于连续段落文字的内容。栏目和高层级内容群组这样的布局结构应该使用div标记，如果是归组一个表单里的多个部分则用fieldset。

3. 引用文字

引用文字可以通过多种方式标记。但是，只有一种（即blockquote，它是一种设计用于长篇引用文字的元素，类似段落那种形式）应该用在基础标记里，因为它在各种浏览器里渲染一致，无需做任何的CSS或JavaScript变通。

引用的内容放在blockquote元素里，来源URI则可以在开始标记中的cite属性里注明：

```
<blockquote cite="http://www.filamentgroup.com">我们为网络应用程序、移动设备和触摸屏终端设计引人入
胜的用户界面，它们不仅简单，而且每个人都能访问。</blockquote>
```

用正确的方式控制分行和空格

自从HTML发明以来，开发者喜欢用分行符和不间断空格（non-breaking space）来控制内容布局和调节间距，而不是用恰当的段落或其他语义标记（以及CSS），因为有时增加一个或多个分行标签更容易。但是，错误使用这些标签可能会严重降低内容的可读性。

**分行（使用
标签）**表示在文本的某个特定位置插入一个硬回车（hard carriage return）。虽然这个元素有它的合理用途，比如给地址分行，但它不应该用来给页面布局增加垂直空间。需要特别注意：在1024×768的屏幕分辨率下看起来不错的分行，可能会在小得多的屏幕（比如手持设备上的屏幕，或者字体放大后的标准屏幕）上显得歪斜，或者留下孤零零的单词。只要有可能，最好让文字自然换行，填满所有可用空间。如果必须精确分行才能正确阅读某一块

内容，比如显示源代码或者诗歌词句时，还可以考虑使用预格式化文本的pre元素。

类似地，**不间断空格**（指定的字符记号是 ，或者字符实体 ）被设计用于连接两个单词，它们中间虽然有一个空格，但是应该位于同一行中，不能独立换行。这个空格有它的合理用途，比如确保专有名词、产品名称或者注册成商标的语句总是一起出现。但是，不间断空格经常被错误用于快速添加元素之间的水平间隔。这种做法就像过度使用分行一样会带来一些风险，比如你的一些用户可能会看见奇怪的空白、孤零零的单词或者别扭的分行。更安全的方法是使用CSS视情况调整外边距或者内边距，以此在元素间制造空间。

需要同时显示引用语和引用来源时，可以使用内嵌于blockquote中的cite元素：

```
<blockquote cite="http://www.filamentgroup.com">我们为网络应用程序、移动设备和触摸屏终端设计引人入胜的用户界面，它们不但简单，而且每个人都能访问。

<cite>ToddParker、Filament集团公司负责人</cite>

</blockquote>
```

HTML4规范还列出了一个内联元素（即q元素），它被设计用于简短的引用文字，比如一句对话：

```
<p>The Donald exclaimed, <q>You're fired !</q></p>
```

理论上，浏览器应当基于网页语言设置或引用文字是否嵌套等上下文背景信息渲染出适当的引号。根据HTML4规范："视觉性用户代理必须确保使用定界引号来渲染q元素的内容"（http://t.cn/zRlYafG）。但是，浏览器实现q元素的方式并不一致，最值得注意的就是Internet Explorer根本不会渲染出q元素的引号。

修复不可预测的q元素

由于q元素的渲染方式相当不可预测，我们建议谨慎地使用它，并用CSS重置（reset）尝试让它在各种浏览器里表现出一致的样式。

推荐读者阅读下列文章，了解让q元素在各种浏览器里样式一致的信息。

"CSS reset and quirky quotes"： http://t.cn/zRlYKlS
"Fixing Quotes in Internet Explorer"：http://t.cn/zRlYNY7

4. 预格式化文本和代码

如果文本已经预先格式化过，必须精确按照原有写法显示，就可以使用pre元素。pre元素通知浏览器要显示出每一个分行或字符空格，而且通常将文字的样式设为固定宽度的字体。它对于显示诗歌、文字艺术图和公式等需要进行特别格式化的内容来说很有用。

```
<pre title="摘自Emily Dickenson的某首诗">
The daisy follows soft the sun,
And when his golden walk is done,
  Sits shyly at his feet.
He, waking, finds the flower near.
```

```
"Wherefore, marauder, art thou here?"
  "Because, sir, love is sweet!"
</pre>
```

在专门格式化代码（比如多行HTML或JavaScript）时，可以将code元素与pre元素一起使用。code和pre两个元素同时使用时会保留内容里的特定分行位置。请注意标记里使用的字符（比如&字符，或者用于开始和关闭标签的小于“<”和大于“>”符号）必须编码为字符实体，这样浏览器才会将它们解读为文本。

```
<pre>
<code>
// 包括下面的脚本块:
&lt;script&gt;
  var el = document.getElementById(&quot;foo&quot;);
  alert(el);
&lt;/script&gt;
</code>
</pre>
```

5. 缩写词

使用abbr元素可以向那些不熟悉缩写术语或首字母组合词的人（以及屏幕阅读器、搜索引擎蜘蛛或者翻译系统）提供即时的解释。要给一个abbr元素添加完整或扩展的文字，可以包括一个title属性，在大多数桌面浏览器里，它会在鼠标悬停在某个单词上方时显示出一条提示。

```
<p>The United States is a member of <abbr title="North Atlantic Treaty Organization">NATO</abbr>.</p>
```

注意　类似的一个元素acronym被用于标识文本中的首字母组合词，不过按照计划它会从HTML5规范中去除。我们建议使用缩写词来代替它：http://dev.w3.org/html5/html4-differences。

---

6. 强调性内容

要强调一个单词或词组可以使用em或strong元素（strong意味着程度更高的强调）。这些元素既可以分开使用，也可以嵌套使用：

```
<p>这个句子使用了<em>多个<strong>强调</strong></em>标签。</p>
```

虽然大多数浏览器默认将em标签里的文字以斜体显示，strong标签里的文字则是粗体，但是如果无需进行额外强调，这些元素就不应当简单用于进行视觉格式化。

如果需要赋予文本较低的内容层级，比如脚注和说明文字，可以使用small元素：

```
<p>人类首次登月是在1964年。<small>来源：Wikipedia，2009</small></p>
```

注意　你可能希望屏幕阅读器在朗读这些元素里的文本时使用不同的强调语气，但是在写作本书时，最新版的流行屏幕阅读器JAWS和Window-Eyes在朗读用这些元素标记的文本时，和普通文本没有任何区别（www.paciellogroup.com/blog/?p=41）。即便如此，用它们给内容里的单词或词组添加语义强调仍然是一种好的做法，而且一旦今后的新版屏幕阅读器增加了这一功能，你已经为它打好了基础。

---

7. 上标和下标

上标和下标字符在某些语言和科学环境中是必需的，可以使用sup（superscript，上标）和sub（subscript，下标）元素进行内联添加。

```
H<sub>2</sub>O
E = mc<sup>2</sup>
```

注意 如果必须大量使用数学表达式或科学符号，请考虑使用一种包含更准确语义标签的标记语言，比如MathML来代替这些元素进行标记。

3.1.2 列表

列表将所列的项目归组到一个容器中，它有三个基本类别。

无序列表呈现一组项目时没有任何隐含顺序，比如一张购物单，在大多数浏览器里的渲染方式是在每个项目前面加一个小点。

```
<ul title="我的购物单">
  <li>牛奶</li>
  <li>鸡蛋</li>
  <li>黄油</li>
</ul>
```

注意 一个好的做法是用无序列表标记导航链接，因为现代屏幕阅读器可以辨认出标记结构里的这些列表，并向用户提供从页面其他区域跳往/跳离此内容的方法。

有序列表包含了优先级或者顺序，在大多数浏览器里的渲染方式是在每个项目前面加一个连续数值。它们可以用于表示目录或者电影队列。

```
<ol title="目录：HTML简史">
  <li>过往岁月</li>
  <li>当前的最新进展</li>
  <li>未来！</li>
</ol>
```

无序列表和有序列表可以互相嵌套，可以将ul元素放进li元素中创建新的层级。

定义列表（definition lists）类似于无序列表，但与为每个列表项分配单个li元素不同，定义列表包含的是元素组：dt（term，术语）和dd（definition，定义）。就像字典条目一样，定义列表里的术语可以有不止一个定义，因此每个dt都允许有多个dd元素。定义列表对于标记功能与描述、姓名与传记或问题与答案这些列表而言是最理想的。

```
<dl title="生于19世纪的著名人物">
  <dt>Abe Lincoln</dt>
  <dd>US president, born 1862</dd>
```

```
  <dt>Albert Einstein</dt>
  <dd>Genius physicist, born 1879</dd>
  <dt>Babe Ruth</dt>
  <dd>Baseball hero, born 1895</dd>
</dl>
```

3.1.3　表格式数据

表格被设计用于显示表格式数据。每张表格必须至少包含table、tr（table row，表格行）和td（table data cell，表格数据单元格）元素才能通过验证，而且在大多数情况下还应该包含th（table header，表格头）标签来区分数据值所在的行或者列。还有一些有用的可选标签可以用来给表格式数据添加语义上的意义，比如使用caption、thead、tfoot和tbody标签标明和归组表格的各个部分。

该选择哪些结构性元素要根据数据的复杂程度，以及它本应如何使用或在视觉上如何显示而定。举个例子，请比较下面两张表格，它们的结构都使用了有效且可访问的标记。第一张表格的形式是最简单的，标记里只包含了列、表格头和单元格：

```
<table>
  <tr>
      <th>State</th>
      <th>Capital city</th>
  </tr>
  <tr>
      <td>Massachusetts</td>
      <td>Boston</td>
  </tr>
  <tr>
      <td>Minnesota</td>
      <td>St. Paul</td>
  </tr>
</table>
```

第二张表格有着复杂的结构，包含了一个caption用作表格标题，内容则进一步划分成表格头（thead）、表格尾（tfoot）和表格主体（tbody）三部分。

```
<table cellspacing="0">
  <caption>Annual Sales:</caption>
  <thead>
      <tr>
          <th rowspan="2" id="department">Department</th>
          <th colspan="4" id="year-2008">2008</th>
      </tr>
      <tr>
          <th id="Q1">Q1</th>
          <th id="Q2">Q2</th>
          <th id="Q3">Q3</th>
          <th id="Q4">Q4</th>
      </tr>
  </thead>
```

```
<tfoot>
    <tr>
        <td colspan="5">Data valid as of December 2009</td>
    </tr>
</tfoot>
<tbody>
    <tr>
        <td headers="department">Toys &#38; Games</td>
        <td headers="Q1 year-2008">1.4M</td>
        <td headers="Q2 year-2008">2.2M</td>
        <td headers="Q3 year-2008">2.8M</td>
        <td headers="Q4 year-2008">2.1M</td>
    </tr>
    <tr>
        <td headers="department">Appliances</td>
        <td headers="Q1 year-2008">1.4M</td>
        <td headers="Q2 year-2008">2.2M</td>
        <td headers="Q3 year-2008">2.8M</td>
        <td headers="Q4 year-2008">2.1M</td>
    </tr>
</tbody>
</table>
```

3

按照HTML规范，tfoot应该紧跟在thead之后（并且在tbody之前），这样网页会先加载表格头和表格尾，然后才是数据行。像这样将一张表格里的各个部分归组进thead、tfoot和tbody元素是很有用的，特别是在组织需要好几秒或者更长时间才能加载的超大数据集时，因为表格头和表格尾元素会更快加载，并向用户提供反馈。

通过将表格头和表格主体的单元格分入不同块，就可以将CSS规则单独作用于thead、tfoot和tbody元素，使它们在视觉上有所区别，或者组合使用CSS和JavaScript给表格主体设置一个overflow属性，这样表格头一栏就能保持固定不动（在基于WebKit或Mozilla的浏览器里只需使用CSS就能实现这个效果，Internet Explorer则需要借助JavaScript才能生效）。

在像上面第二个例子这样相对复杂的表格中，一个单独的表格头横跨了多列或者多行。从浏览器的呈现方式看，表格头和数据单元格之间的关系可能显得很明确，但是当屏幕阅读器等非视觉性用户代理解读标记时，这种联系可能就没那么明显了。为了确保使用辅助技术的用户不会错失宝贵的数据联系，可以给每一个th指定一个独一无二的ID，为每个单元格添加headers属性，然后把表格头ID写入headers属性，将每个单元格映射到它对应的表格头上。每个headers属性可以指向多个ID，ID之间用空格分隔。

还可以使用caption元素添加表格说明，或者提供用户解读数据可能需要的额外信息。caption元素应当在开头的table标签之后紧跟着写进表格结构中，位于表格主要内容的前面。默认情况下，表格说明直接显示在表格上方，就像是个标题。

注意 table标签里的summary属性和表格说明大体上是重复的，所以在HTML5规范中被去掉了。因此，我们建议在需要额外添加帮助解释信息时使用caption，而不是summary。

在增强体验里，表格式数据可转换成图表、曲线图和其他可视化形式。请参见第12章了解如何实现这些转换。

终于不需要用表格布局了

在CSS成为绝大多数浏览器支持的网络标准前，开发者们通常依靠表格来布局网页内容。他们经常会制作复杂嵌套的表格来格式化页面里的每一个内容块，甚至还引入了“空白间隔”图像来模拟内边距和外边距。

既然现在CSS已经被广泛使用，就不再需要这样依赖表格了，可以让它们回到原本的用途上：显示表格式数据。我们从来不推荐用表格实现布局。

3.1.4　图像

img标签将图像嵌入网页中。因为图像除了源文件名之外不包含语义上的意义，所以很重要的一点是要用alt属性提供描述性的文字给屏幕阅读器、搜索引擎和禁用图像的用户，使他们能够理解图像传达的内容。

```
<img src="acme-logo.png" alt="Acme公司" />
```

以下是填写alt值时需要注意的一些原则。

- 图像缺失或不可见时，可显示或读取alt属性的值，因此可以将alt文本看作是图像的替代，而不仅仅是一段描述。要考虑图像在网页中的作用，以及它的内容对理解页面大意而言是至关重要还是可有可无。
- 任何在网页标记中纯粹起装饰作用的图像（比如某种设计花纹）应当包含一个空白的alt属性，类似于alt=""，屏幕阅读器就会朗读文件名并继续下去。要尽一切努力限制标记中内嵌的纯展示性图像的数量，而且只要有可能，用CSS将它们指定成背景图像。
- 图表和曲线图的alt文本应当简要描述任何有关的结论、趋势或数据关系。一段内容为“2009年总销售量折线图”的alt描述远不如“2009年度的总销售量曲线图显示出18%的增幅，从1020万增至1210万（美元）”有用。
- 照片和插图只有在它们是真正的内容，而且对网页大意至关重要时才应当被详细描述。不要把alt当成是视觉图像的说明文字，例如“张三的肖像”。但是别忘了描述能给网页大意增添背景因素的有用信息，例如“这张肖像属于20岁时的张三，一个脏兮兮、披头散发的牛仔，身着皱皱的白衬衫和牛仔裤，观察着他蒙大拿农庄里的崎岖地形”。
- 没有必要将“图像”一词包括在描述当中，因为屏幕阅读器一般会把role和文本一起朗读出来。举个例子，要使用alt="一匹马和骑手"，而不是alt="一匹马和骑手的图像"，因为它可能会被读成“一匹马和骑手图像的图像”。

3.1.5 嵌入式富媒体

在网页中嵌入富媒体的最佳做法一直以来都有争议，它们经常改变，以适应私有内容分发之间的竞争格局。富媒体最常见的网络分发技术需要用到浏览器插件，例如Adobe Flash、苹果的QuickTime、Sun Java或者微软的Silverlight，而且嵌入页面的方式也有好多种，比如使用object元素，或者embed等非标准元素，很多时候还会同时使用这两个元素。

3

插件虽然很流行，但却存在着很多缺点。

- 对用户来说，为了浏览网页从而不断下载和升级插件可能是件很烦人的事；对开发者来说，随着插件的发布、升级或者终止，富媒体内容需要不断维护。
- 插件的格式将内容封装起来，这可能会导致内容难以被搜索引擎、屏幕阅读器和移动设备访问到。残障用户经常会被完全排除在富媒体内容之外，原因是提供商认为通过某一特定插件分发可访问的内容花费过高，或者过于复杂，有时则是技术上不可行。
- 内容的分发依赖于插件制作公司对其产品的持续支持，这可能会让那些希望确保内容持久性的人感到担忧。

所有这些原因导致人们对一种标准化富媒体分发系统的需求空前强烈。幸好，HTML5规范引入了几种可用于嵌入式富媒体的元素（video、audio以及用来生成图像和动画的canvas），它们简单、可访问并且面向未来。

HTML5的video元素可以将视频内容嵌入至网页中，方式和添加图像几乎一模一样。只需简单地将video标签插入标记中，把它的src属性指向某个视频文件的URL，然后支持此功能的浏览器（就目前而言，需要最新版的Safari、Firefox和Opera）就会向用户提供一套原生界面，供他们观看视频和对播放过程进行控制。

```
<video src="myvideo_320x240.ogg" />
```

HTML5的audio元素与video元素很相似。当然，区别在于它嵌入网页的是一个音频文件。只需简单地将audio标签插入标记中，然后把它的src属性设置成某个音频文件的URL即可：

```
<audio src="myfavoritesong.wav"></audio>
```

另外，audio和video元素两者都拥有能控制播放过程的原生JavaScript API，可以借助许多事件，创造出能与任何基于插件的实现方式相媲美的交互性。这些API记录在w3.org网站上的HTML5规范里。

HTML5为图形富媒体（例如交互性图表或功能图形动画）提供了canvas元素，你可以把它看做一种图像元素，但却内含一套用于绘制图形的JavaScript API。（第12章会提供一个例子，展示如何将表格式的HTML数据转换成基于canvas的图表，并能在所有主流浏览器里显示。）

这三个元素都支持在开始和结束标签之间放置标记，可以用来在不支持HTML5标准的浏览器里引用内嵌媒体的替代图像或文本，或者提供操作说明或链接，以便用户访问来源。

```
<video src="myvideo_320x240.ogg">
哎呀，你看到这条消息是因为你的浏览器不支持<code>video</code>元素。要观看这段视频，
```

```
请<a href="myvideo_320x240.ogg">下载 myvideo_320x240.ogg</a>，然后在别的应用程序里播放。
</video>
```

虽然HTML5规范仍然在审议当中，但我们强烈建议现在就用这些元素嵌入富媒体，代替原来的私有插件，因为它们的语义更为丰富（每个开始标签都标明了内容的类型和来源），而且更重要的是，它们支持添加替代性标记以兼容不支持这些元素的浏览器。如果必须为别的浏览器提供另一种富媒体作为替代品，你可以（举个例子）写一个脚本检测浏览器是否支持video元素，如果不支持则插入另一种元素（比如object），将浏览器指向它能够读取的视频文件格式。

3.1.6　嵌入外部网页内容

在Ajax引入无需整体刷新就能动态更新部分页面区域的能力之前，开发者们使用frame和iframe实现类似（不过没那么复杂）的功能。

frame和iframe元素都被设计用于将外部内容引入网页，它们加载整张HTML网页及其所有附属内容。frame元素从属于一个更大的元素群组框架，这个名为frameset的框架取代了body标签，将网页分割成数个可能相互关联或者依赖的区域。iframe元素则被设计用于内联插入，与其他标记平等相处。

frame和iframe元素带来的灵活性（可用于建设极度复杂的网站，内含深度嵌套的框架组来模拟服务器端常见的复杂依赖结构）会导致严重的可访问性、导航和页面性能问题。因此，frame、frameset和noframes元素在HTML5规范中被去掉了，原因是“使用它们会对终端用户的可用性与可访问性造成负面影响”（www.w3.org/TR/html5-diff/#absent-elements）。

谢天谢地，用嵌套的frameset来构建页面的日子早已过去。但是，即使在今天也可能会出现一些情形，导致必须出现在页面里的内容无法包括在页面标记中。这也许是因为内容被托管在外部网站上，也许是因为内容的格式会造成视觉冲突，所以不能直接插入网页。在这些情况下，你可能使用iframe呈现此内容（不过我们建议尽量不要这么做，因为这种方式存在性能与可访问性方面的缺点）。

在那些适合使用iframe的场合，我们建议编写基本版网页时使用一个指向资源的锚链接（anchor，即a元素）并包含一段文字描述，然后再用JavaScript生成并插入一个iframe元素，在它的src里填入锚链接元素的href属性。

举个例子，我们会在用于输出到所有浏览器的基础HTML里写入一个标准的锚链接标签：

```
<a href="http://filamentgroup.com" class="inline-content">我最爱的设计公司</a>
```

如果能力测试成功通过，就用锚链接的href属性创建一个iframe，在网页里内联显示整个网站：

```
<iframe src="http://filamentgroup.com"></iframe>
```

在网页里插入iframe后，可以把锚链接隐藏起来，也可以让它留在增强体验里，以便用户直接访问该网站：

```
<iframe src="http://filamentgroup.com"></iframe>
<a href="http://filamentgroup.com">Filament集团的网站</a>
```

3.2 标记交互内容

交互性是万维网的一个基本特征，而语义标记提供了众多的选项来支持一系列具体交互类型，从简单的链接到复杂的数据输入表单不等。

原生HTML交互元素的内置功能很丰富：它们帮助实现跳转；捕捉和提交数据，并显示反馈；为各种浏览器和设备环境下可行的键盘和鼠标交互提供支持。HTML的表单元素提供了捕捉用户输入的首要方式，以及将其提交回服务器进行存储或处理。

考虑如何构建包含花哨的CSS和JavaScript增强交互组件的网页基础标记时，理解原生交互元素的语义意义和独特能力显得尤为重要。与增强体验的代码不同，基础标记必须不依赖CSS或JavaScript的协助也能正常工作（使用户有机会完成所需的全部任务）。

前一章描述了X光透视，它是一种将复杂设计分解成基本却具备功能的多个部件的方式。例如，观察到某个自定义的下拉菜单其实只不过是美化过的select元素，或者某个滑块其实是两个文本输入框的化身，用于捕捉一系列数值。本书第三部分会详细讨论一些增强交互组件，并为它们的基础标记寻找最佳的语义选择。

现在，我们将探讨交互和表单元素的整体结构，它们的主要用途，以及考虑将每一种元素用于基础标记时，需要加以研究的所有重要事项。只要有可能，我们会注明某个元素在HTML4和HTML5规范中的变化。我们还会列举将其升级用于增强体验的各种可能方法。

3.2.1 锚链接

锚链接是万维网里的基本导航元素，用于链接网页以及链接到同一页中的内容。

要链接到同一个网站里的另一页或者某个外部网站，应在链接的href属性中指定一个目标URI：

```
<a href="http://www.acmecorp.com" title="Acme公司官方网站">Acme</a>
```

要链接到同一个网页内的某个位置，应指定该位置以#开头的唯一id：

```
<a href="#content">了解我们做过些什么</a>
...
<div id="content">
  <h1>我们做过些什么</h1>
  <p> 我们曾设计过种类繁多的网站、应用程序界面、品牌化和印刷项目，涉及各种行业。</p>
</div>
```

在增强体验里，我们可以做很多厉害的事情来改变锚链接。举个例子，可以使用title属性或者本地锚链接，生成在鼠标悬停时显示的工具提示。外部链接的URL可以用JavaScript进行“绑架”，窃取某个链接上的鼠标点击事件后，发起一个基于href值的Ajax请求。

3.2.2 表单结构

任何网上交易或者数据输入方案都是用表单元素实现的。结构良好的表单包括用数个基本语

义元素布局工作区，再用一个或多个表单控件来收集用户输入并提交数值以供处理。

1. form

每张表单都需要的第一个元素当然就是form元素，它的action属性指向某个会在表单提交后对其进行处理的资源：

```
<form action="edit-account.php">
  ……表单元素放在这里……
</form>
```

注意　HTML5规范中声明“input、output、select、textarea、button和fieldset元素里的新form属性允许控件关联到一张表单。也就是说，这些元素现在可以放置在页面中的任何位置，而不仅仅只是作为form元素的从属”（www.w3.org/TR/html5-diff/#new-attributes）。鉴于已经有更多的浏览器支持了这一规范，我们可以利用这个新的属性。不过目前来说，我们建议将所有应当随单个form元素一起提交的表单控件都放在该标签内，从而兼容所有浏览器。

---

2. fieldset和legend

fieldset元素是可选的，可以用于将表单元素组织成不同的逻辑区域。举个例子，联系信息、账单信息、一对用户名和密码或者一组单选或多选按钮可以归组到一起，共享同一个标题。legend元素位于fieldset元素内部，是群组的标题。

```
<form action="edit-account.php" method="post">
  <fieldset>
    <legend>联系信息</legend>
    ...
  </fieldset>
</form>
```

注意　HTML5规范允许使用fieldset元素的disabled属性，因此“设置该属性会禁用其中的所有内容”（www.w3.org/TR/html5-diff/#new-attributes）。

---

3. label

每个表单控件都应该有一个label元素来标识它，并提供一种额外的方式让这个控件获得焦点。正确关联标签和控件后，点击某个标签会自动将焦点转移到它的控件上，这就意味着你可以通过点击标签选择一个复选框。这是一种创建更大点击区域的理想方式，让表单对那些视觉或运动控制机能受损的用户来说更具可访问性。

label和其元素关联的方式是指定一个for属性，将值设为该控件的ID：

```
<label for="filling-1">花生酱</label>

<input type="checkbox" value="pb" name="sandwich" id="filling-1" />
```

标签及其元素之间的这种关系对复选框和单选按钮来说也很有用。可以借助共享的点击事件自定义标签，比如创建一个投票组件，就像Netflix上的星级评分一样。（请参见第15章里的范例，了解如何实现。）

某种设计需要不带可见标签的表单控件时（比如一个地址字段可能由两个文本输入框叠加而成，但两者只有一个标签），我们主张为每个输入框都创建一个标签，然后仅隐藏不该显示的那个标签：


```
<label for="address-1">地址</label>
<input type="text" id="address-1" />

<label for="address-2" class="hidden-label">地址, 第2行</label>
<input type="text" id="address-2" />
```

第一个标签“地址”在页面上可见，用户使用Tab键跳跃到此字段时会获得焦点；第二个标签“地址，第2行”出现在标记里，但以一种可访问的方式隐藏起来，因此屏幕阅读器仍然能够看到。要以可访问的方式隐藏第二个标签，应设置一个较大的`left`值，将其绝对定位于页面之外：

```
.hidden-label { position: absolute; left: -9999em; }
```

3.2.3 表单控件

语义标记包含了一系列表单元素，或者说控件。每一种都受到业务规则的支配，后者定义了它们各自的能力（包括其数据选项的具体结构和格式，是否支持单选或者多选，各种选择是受到系统限制还是可以由用户定义），以及丰富的原生交互行为。这些元素为可靠的交互奠定了坚实基础，同时还为在增强环境下进行转变提供了众多可能性。

1. 文本输入框

最简单同时也可能最常用的表单元素就是文本输入框，它接受单行字母数字（alphanumeric）文本：

```
<input type="text" name="address" />
```

考虑许多组件的基础标记时，文本输入框都是理想的候选表单控件，因为它的用途如此多种多样，可以捕捉字母数字字符的任意组合。

HTML5提供了许多输入类型，从而更严格地验证文本输入框，例如`number`、`telephone`、`range`等。我们建议，只要能用上，就应该在基础标记里使用这些输入类型，因为浏览器不能识别这些特殊类型时，会仅使用默认类型`text`。

举个例子，下面的标记展示了一个`type`为`range`的输入框，它包含了`min`和`max`属性以说明可接受的值：

```
<input type="range" min="0" max="50" value="25" />
```

`range`这种输入类型在不同的浏览器上会有不同的外观：Safari 4和Opera 10将这个输入框渲染成一个滑块，而Firefox 3.5等其他浏览器只是简单地将其渲染成一个文本输入框，如图3-1所示。

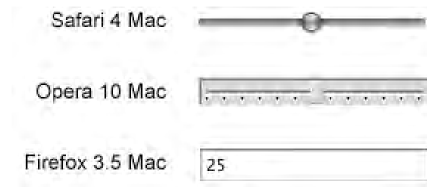


图3-1　多种浏览器里的range输入类型（上图：苹果Mac版的Safari 4。中图：Mac版的Opera 10。下图：Mac版的Firefox 3.5）

最新版的Opera浏览器（Opera 10）对这些输入类型提供了最详尽的支持。如果将类型设为number，Opera 10会给输入框渲染出“微调”（spinner）箭头，方便用户使用鼠标或方向键增加或减少数值，如图3-2所示。类似地，date类型会将输入框渲染成一个附带自定义日历控件的下拉菜单。



图3-2　在Opera 10浏览器里用输入类型date格式化年份

（第16章会使用输入类型number来操作某个自定义滑块的值。）

新的placeholder属性同样来自于HTML5，它可以应用在文本input元素（以及textarea）上，在字段内部显示出提示文字，告知用户它期望的特定格式，如图3-3所示。

```
<label for="groceries">日用百货：</label>
<input type="text" name="groceries" id="groceries" placeholder="添加日用百货项，各项之间用逗号分隔" />
```



图3-3　Safari 4浏览器里带有placeholder属性的文本输入框

开发者为了实现这种效果，通常会预先生成提示文字并填入value属性，这种做法会导致可访问性问题，而且需要加以验证，以防提示文字的值随表单一起提交。在那些不支持placeholder属性的浏览器里，可以使用JavaScript来操作文本框里的文字，以模拟原生的实现方式。

2. 文本区

设计textarea元素的目的是输入多行文本，常用于输入礼品留言或者评论等。textarea元素有一个开始标签和一个结束标签，标签之间包含的任何内容都会显示在一个可编辑的文本框内。虽然你可以用CSS控制它的宽度和高度，但是还应该考虑添加cols（宽度）和rows（高度）属性，设置textarea在基本体验里的总体尺寸：

```
<textarea cols="30" rows="10">
Four score and seven years ago, our fathers brought forth upon this continent a new nation...
</textarea>
```

在某些有能力的浏览器中，textarea可转变成一个富文本编辑器，或者如果textarea里有一个各项由逗号分隔的列表，那么它也可以增强成一个接收者列表生成器，就像Facebook的消息接收者文本区那样。（第18章会演示如何将一个textarea转变成列表生成器。）

3. 复选框

checkbox元素的功能是让用户从一组选项中选择一个或多个。默认情况下，所有选项都会内联显示在页面中，可以通过鼠标点击，或者在某个选项框获得焦点后按下空格键来切换选择状态。还可以通过添加checked属性，使复选框默认处于选中状态，该属性只接受一个值：checked。（很好记吧？）

要让复选框正确提交，一组复选框里所有选项的name属性必须相同：

```
<label for="filling-1">花生酱</label>
<input type="checkbox" value="pb" name="sandwich" id="filling-1" />

<label for="filling-2">果冻</label>
<input type="checkbox" value="j" name="sandwich" id="filling-2" />

<label for="filling-3">蜀葵糖霜</label>
<input type="checkbox" value="m" name="sandwich" id="filling-3" />
```

复选框可以增强成帮助性的组件，比如自定义切换按钮。因为复选框是出了名地难用CSS打造一致的跨浏览器样式，所以我们开发了一种方法来自定义复选框和标签对，使它们能够实现某种自定义的增强体验设计。第15章会简要介绍这种方法。

注意 事实上，大多数浏览器会将空的checked属性（checked=""）解读成该复选框已被选中。它们会忽略空的引号，只看checked属性本身是否存在。checked属性原本被设计成一个布尔型（Boolean）选项，可以附加到form元素里：<input type="text" checked>。随着XHTML的出现，这一语法相应发生了变化（变成了checked="checked"）以遵循XHTML的语法规则。HTML5允许使用两种语法中的任意一种。

使用JavaScript切换选中状态时尤其要注意这一点。通过DOM操纵选中状态（this.checked = true）比切换checked属性的checked值更可靠。

---

4. 单选按钮

单选按钮的作用与复选框非常相似，不同之处在于，单选按钮只允许选中一组互斥项目里的其中一项，而复选框则允许多重选择。

基于这个原因，单选按钮的理想用途是标记那些需要从一小组选项（比如说最多10个）里进行单选的界面，以及某些适合同时在屏幕上显示所有选项的情形，比如多项选择题测试或调查问卷中的问题，评分或投票，或者某个零售结账流程中的发货选项。和复选框一样，单选按钮可以使用checked属性预先选中，但是一组按钮里任一时刻只能有一个选中项。

与复选框的相似之处还有，单选按钮很难用CSS修改样式，不过同时使用CSS和JavaScript就可以很容易地将其升级，从而用于某种增强体验。第15章会讨论几个案例，里面将一些单选按钮增强成单个滑块组件（比如音量控制器）或者星级评分组件。

5. 下拉列表框

select元素允许用户从一张预先定义好的列表中单独选择一个选项，就像单选按钮一样。因为原生的select元素被渲染成只显示当前选中的选项，而非整张列表，所以它的理想用途是那些必须塞进紧凑空间里的大型列表。select在激活后会展开一张菜单，列出每个option元素：

```
<select name="north-american-countries">
  <option value="canada">加拿大</option>
  <option value="mexico">墨西哥</option>
  <option value="united-states">美国</option>
</select>
```

提示　虽然一个select理论上能容纳的选项数量非常大，但是出于可用性考虑，最好将这一数字限制在大约50以内，以避免滚动列表过长。如果超出这一数字，不妨使用optgroup元素将条目分组，前提是此内容适合分成子群组。在任何情况下（哪怕选项列表只包含8~10个值），将选项以用户能理解的逻辑顺序排序后再呈现出来，都是一种体贴的做法，能使列表易于浏览和滚动。

每个option元素都包含一个单词或词组，显示在select的下拉菜单里，另外还有一些属性，可以在提交表单或将某个基本select元素转变成增强体验里的组件时使用。

- value——随表单一同提交的值（必需属性）。
- selected——可以设置为selected，从而在页面加载时显示出选中的option。如果允许多选，那么可以有多个option具备这个属性。
- disabled——使选择菜单里的某个选项可见但不可选。
- label——该选项所含文本值的某种缩写。（此属性是合法的，但不是所有浏览器都能很好地支持。因此，不建议将它用于基本体验所需的关键信息，不过如果要编码额外的意义，从而在增强组件里使用，那么它非常有用。）

下拉列表框里的option还可以通过optgroup元素分组到子群组中。虽然这是合法的标记，但是optgroup在各种浏览器里有不同的显示方式，结果导致无法保持一致的样式。虽然CSS支持有点受限制，但是我们仍然推荐在基础标记里使用optgroup，为列表提供额外的语义意义。

可同时选中多个选项，方法是将select元素的multiple属性设置为multiple（类似于设置文本输入框的checked属性），不过不推荐使用这一功能，因为没有视觉指示器表明可以选择多个项。用户必须知道要在按下Control或Shift键的同时点击才能进行多选（或者你必须提供一段操作说明文字才能达到目的）。这种情况下，我们认为一组复选框更直观，而且更可用。

如果设计要求select元素或它的菜单具备自定义的外观（比如在每个选项旁边显示图标，或者用自定义的版式高亮显示部分文字），可以从将select元素作为基础标记着手。请参见第17章，

了解如何用这种方式制作自定义下拉列表框。

6. 用input和button提交表单

HTML提供了两个可以提交表单数据的元素：一个是type设置为submit的input元素，另一个是button元素（type同样是submit）。input的value属性作为按钮文本显示，而button显示的是标签内包含的文字。下面两行代码会渲染出两个外观相似的按钮：

```
<input type="submit" value="保存更改" />
<button type="submit">保存更改</button>
```

提交input和button的目的相同：它们正确包含在form标签中时，点击它们会将此表单的数据发送到服务器。

它们还可以很方便地用CSS修改样式。不过，button元素允许嵌套子元素，这就意味着它相比input能支持更多的高级CSS样式定义，包括文本层级和“滑动门”式的背景图像。

在Internet Explorer里提交按钮元素的值

大多数浏览器以相同的方式处理表单提交，无论用户点击的是input还是button元素。但是，当input和button的value属性必须和数据一起发送时，某些版本虽老却依然很流行的Internet Explorer（包括IE 6和IE 7）存在着异常情况。在IE 6或IE 7里点击按钮后，button元素的innerHTML（也就是“保存更改”）被发送给服务器，而不是指定的value。IE 8修正了这个问题，但是在旧版本被淘汰之前，有必要使用一个JavaScript变通方式，在所有浏览器里正确提交button的value。

要想了解更多信息，请参见“Coping With Internet Explorer’s Mishandling of Buttons”（http://t.cn/zRlYk8P）。

在基础标记里，建议使用input来提交，因为它是HTML的核心创始元素之一，历史可以回溯到HTML规范最早的完整版本之一：1995年发布的2.0版。作为一种更加古老的元素，input更有可能会在老式或小众浏览器上正常工作，或者在那些只部分支持当前HTML规范的浏览器上正常工作。相比之下，button元素则比较新（在HTML规范的最新完整版，即4.01版里才被引入），老式移动设备对它的支持偶尔会不太可靠。虽然button在大多数浏览器里应该都能用，但是更安全的选择是在基础标记里使用input元素。

如果你的设计需要一个自定义样式的按钮，且包含多个HTML元素，那就应该用JavaScript将input转变成button。第14章将介绍了这一技巧。

3.3 创建页面环境

本章之前描述的这些语义标签，提供了正确构建网上显示的最常见内容类型所需的工具，但是这些单独的部件通常还需要一层组织架构，以界面容器或群组的形式让它们在网页环境中具备意义。关键之处在于，你要做一些简单但却重要的选择，确保内容以符合逻辑的方式分组，以此向使用你内容的各类人群和用户代理传递最适当的语义意义，并以一种能提供最高效和最令人满意的用户体验的方式来呈现它。

3.3.1　了解何时该用块级元素或内联元素

块级（block-level）和内联元素在页面结构里有不同的作用，它们的默认显示属性也不同，可能会影响基础标记在基本体验里的渲染方式。理解这一区别很重要，它关系到你能否为网站做出正确的选择。

设计块级元素（包括标题、段落、块引用文字、列表、表格和div）的目的是组成页面里高层次的结构和内容层级。默认情况下，块级元素在页面中垂直堆叠，还可以同时包含块级和内联的子元素。内联元素（比如锚链接a、增加强调的元素em、strong和small，以及span）则名副其实：它们用于在块级容器内与文本内联流动（flow）和绕排（wrap）。

提示　编写网页基础标记时，请考虑块级和内联元素的默认显示属性对无CSS或JavaScript页面的渲染方式的影响，从而帮助改进页面格式。

用HTML5元素归组内容

一些HTML5元素用来代替div以实现特定的布局组件，包括section、aside、nav、article、header和footer。虽然它们在语义上比对应的通用元素更为准确，但这些向前兼容的HTML5元素可能得不到旧版或只具备最基本功能的浏览器的支持，这就意味着它们可能不会遵守面向它们的CSS规则。在浏览器支持得到改善之前，这些元素应该只用来增强基础标记，而不应成为它的基本组成部分。或者你可以考虑将它们绕排在现有的结构性元素周围，这样既可以利用它们的语义优势，又不用冒在不支持它们的浏览器里出现问题的风险。举个例子，一个用作页头的div元素可以用header元素包围起来，或者相反：

```
<header><div id="header"></div></header>
```

另外，这些元素中的一些所表达的语义意义光从名字上看不太出来。举个例子，section就像div一样是一种块级元素，用来归组页面里各种类型的内容，然而，它还有另一项重要功能：每个section都代表了页面“大纲”里的一个节点。如果你把某个文档里所有的标题和内容转换成大纲形式，那么每个section都会成为这个层级结构里占据一个新行的条目。

要了解这些元素的更多信息，我们推荐A List Apart的这篇文章：“A Preview of HTML5”（www.alistapart.com/articles/previewofhtml5）。

大多数语义元素的本意是描述特定的内容部件，比如标题和段落。但是，div和span元素的定义却不那么明确。两者的主要用途都是在缺乏更能胜任的语义元素时归组内容。

div是一种块级元素，通常用于构建布局组件，比如页头、栏目和页脚。它还可以用于构建缺乏HTML语义等同元素的页面组件，比如某个日历组件的结构。举个例子，可以用div创建一个多栏布局，然后使用标题和段落元素来组织某篇新闻文章的内容结构：

```
<div>
  <p>我是一个位于div内部的块级段落。</p>
</div>
<div>
  <h3>文章标题</h3>
  <p>我是一个位于div内部的块级段落。</p>
</div>
```


span则是内联使用，理想的用途是归组文本行里的词组。内联元素只能包含其他有效的内联元素。

```
<p>产品策略，<span>以用户为中心的工作流设计</span>，概念演示和UI原型……</p>
```

3.3.2 用ID和类标识元素

id和class属性可以应用在大多数HTML元素上。类和ID让元素更容易被CSS和JavaScript找到，同时还有其他一些有用的作用。

一个ID在页面里必须是唯一的，也就是说它只能在单个元素上用一次，比如页头、页脚或者特定的表单控件。ID属性还有一些其他用处。

- 作为某个锚链接的本地目标。用户可以点击一个指向某个特定ID的链接（以#开头：`<a href="#table-of-contents">目录</a>`）将页面滚动到那个元素，或是创建一个书签供将来引用。
- 让元素之间形成一种有意义的关系，比如将某个label元素连接到它所描述的表单字段上，这样用户点击标签就能让这个字段获得焦点。
- 让使用辅助技术的用户能理解元素间的关系，而这些关系可能只对视力正常的用户来说很直观，比如某个增强版树形控件里的当前活动节点，或是某个菜单按钮和它的下拉菜单主体之间的关系。

与ID不同，类用于标识在页面或网站里重复出现的元素，因此它们可以很方便地将CSS或JavaScript同时应用到多个元素上。除了应用增强信息的功能外，类还在*微格式*（microformat）中用于描述网上常见的标准内容类型，例如某人的联系方式或者某个活动的日期和地点。虽然微格式不是W3C官方的Web标准，但它已经吸引了不少人，一些网站和浏览器借助它们的语义，以有趣的方式连接分散在网上各处的内容。（要了解更多信息，请访问microformats.org。）

无论你出于什么原因将类或ID添加到标记里，都建议你选择能够准确描述元素内容和用途的名称，以便开发者理解标记，而不是把它和某个具体的视觉设计或行为特征捆绑到一起。如果设计发生了改变，哪怕只是一些细节，基于设计的类名和ID名就可能会迅速变得毫不相关而且令人困惑。为了实现这个目标，你还应该在网站里自始至终使用一致的命名惯例（比如，使用连字符来连接单词），保持代码的整洁和可预测，从而简化代码维护和延伸（假如网站需要扩展）工作。另外，因为div和span都不传达它们所包含内容的语义意义，所以指定描述性的类名和ID名来清楚标明它们在代码里的作用就显得尤其有用。

3.3.3 用WAI-ARIA路标角色标识页面主要版块

WAI-ARIA（Accessible Rich Internet Applications，可访问富互联网应用）套件是一个W3C规范，其中包括了一些建议，用来给动态和高级用户界面控件所用的标记赋予意义。ARIA套件还提供了一些被称为路标角色（landmark role）的属性，可以指派给基础标记里的HTML元素。

路标能标识出网页的主要版块：页头、页脚和主要内容等。新版的屏幕阅读器让用户可以通过按键（比如JAWS上的分号键）在路标之间导航。负责标识静态内容版块的路标角色包括：

- banner——用于一个在网站里始终显示的全局性页头；
- complementary——用于支持性内容，比如侧边栏；
- contentinfo——关于此网页内容的信息，比如脚注或版权声明；
- main——用于页面里的主要内容，比如一篇新闻文章的主体部分；
- navigation——用于导航链接列表；
- search——用于搜索表单。

可以使用role属性将路标指派给页面的某个部分：

```
<ul role="navigation">
  <li><a href="home.html">首页</a></li>
  <li><a href="about.html">关于本公司</a></li>
  <li><a href="contact.html">联系我们</a></li>
</ul>
```

虽然ARIA是一个相对较新的规范（对它的支持仅限于新近版本的浏览器和屏幕阅读器），但是ARIA的属性可以在不能理解它的浏览器中安全使用（它们会被忽略掉），只需要很少的工作就能在今后给屏幕阅读器用户带来巨大的好处，因此我们建议现在就在你的项目里部署它们。（第二部分会讨论如何将ARIA属性指派给由JavaScript动态生成的标记。）

3.3.4 保持源代码顺序清晰易读

源代码顺序指的就是将标记写入某个HTML文档的顺序。要让网页通过验证，一些元素必须出现在其他元素之前：举个例子，head元素必须出现在body元素之前。但是，在body标签内部，特定页面组件的代码块应出现在标记里的什么位置由开发者决定。主内容栏应该排在侧边栏的前面，还是相反？这很重要吗，它们不都是用CSS浮动到相应的分栏里的吗？事实上，这对那些不依赖视觉线索阅读网页的用户而言*确实*很重要。标记的排序方式可能会对他们造成巨大影响。

以一种有意义的顺序从上到下组织标记，并在整个网站里贯彻应用这一逻辑，这对使用不完全支持现代CSS方法的浏览器（包括那些使用旧版浏览器或者某些移动设备）以及类似屏幕阅读器等辅助技术的用户而言，是让你的网站清晰易读的关键。经过组织的标记能表达内容层级，并提供一条清晰的导航路径，而不统一、胡乱堆砌的标记会阻碍对内容的理解和导航。

依靠CSS来承担组织内容的艰苦工作可能很吸引人，因为它提供了无数的方式来调整位置，以及用图像、颜色和字体大小（仅举几例样式特征）突出重点内容。但是，编写不附带CSS的标

记能够确保它对所有用户都清晰易读，并且降低了一种可能性，即虽然网页看上去组织良好，但它的内部其实是一个让人困惑的标签迷宫。为了保证安全以及标记源代码顺序清晰，请坚持原则，只在标记的源代码顺序完整就位后才使用CSS。

来做个有趣的测试，选择一个你喜爱的网站，然后在开发者工具栏选项里彻底关闭CSS（在Firefox的插件Firebug里，禁用所有CSS是CSS菜单里的一个选项），或者注释掉你自己网页里的样式表引用，看看结果如何。看到了什么？将你的结果与下述情况比较一下。

3

- 最重要或最及时的内容处于页面顶部，重要性较低的内容位置也较低，处于网页中线以下。
- 内容的分组很清楚，都有标题予以标明，其后是相关的段落。
- 标题正确传达了内容的层级：优先级越高，标题字体越大（比如h1或h2元素），其后则是优先级较低的元素。
- 链接处于正确的上下文环境中（举个例子，一个名为“更多……”的链接应该位于一排缩减版链接列表的末尾）。
- 内嵌图像出现在文字说明之前。
- 全局导航元素在每张网页里都处于相同的位置，而且不会把主要内容过度推往页面下方。

如果你的结果与上面这些情况相符，那么源代码顺序的组织方式就是正确且可访问的；如果不符，那些使用旧式、移动或屏幕阅读器浏览器的用户就可能无法理解你的网站。用户的黑莓手机载入网站时，他们首先应该看到优先级最高的消息，如果他们不得不滚动屏幕才能找到它，那么这个消息（还有这个机会）就可能会被错过。

全局导航的摆放位置和“跳过”链接

在源代码顺序里摆放全局导航元素是特别需要技巧的。全局导航中的链接数量是需要考虑的一个重点，因此在选择摆放全局导航标记的位置时，我们会遵循一套通用的规则。

- 如果导航链接的总数量相对较小（包括主链接和次级链接在内的可选项共有10~25个，或者更少），我们就让它处在页面顶部，大多数用户（甚至包括屏幕阅读器用户）都希望在这里找到导航。
- 如果导航规模比较大（特别是多级导航菜单），就在源代码顺序里把导航标记放在内容后面。

网页顶部一张非常小的列表可以很容易通过滚屏越过，如果提供了链接也可以用它跳过。长列表则可能会把更有价值的独特内容推向页面下方，因此用户必须滚屏才能找到。如果你预计用户可能会在非常小的移动设备屏幕上浏览网页，那就需要考虑滚动长长的选项列表才能找到好东西有可能会让人感觉烦闷，更糟糕的是可能会阻止用户继续阅读下去。

把导航标记放在主要内容区域之前还是之后都没有给它们一个固定的家更重要。如果导航在不同页面源代码顺序里的位置不同，那么屏幕阅读器用户就不得不重新摸索它的位置，并可能要忍受再次听读导航选项的不便。

无论是哪种情况，我们都主张要采用添加“跳过导航”锚链接这一良好做法（或者当导航在页面底部时添加“跳至导航”链接，方便定位），让键盘或移动用户向前跳至主要内容，并避免在访问每一页时都要重复这些选项。

在基础标记的源代码顺序里，跳过链接应该紧挨着写在它所管控的内容之前，像这个重复出现在每一页上的顶部导航列表：

```
<a href="#content" id="skip-navigation">跳过导航</a>
<ul id="nav" role="navigation">
  <li><a href="home.html">首页</a></li>
  <li><a href="products.html">产品</a></li>
  <li><a href="contact.html">联系我们</a></li>
</ul>
<div id="content">
  <!-- 这里是网页内容 -->
</div>
```

这个链接在增强体验里可以用CSS隐藏起来，不过要确保以可访问的方式隐藏它，比如将它的位置移到页面之外。如果使用display或visibility属性来隐藏链接，屏幕阅读器用户就无法使用它。

```
a#skip-navigation { position: absolute; left: -9999em; }
```

对于使用键盘的视觉用户来说，良好的做法是给链接设置样式，让它在获得焦点或者处于活动状态时显示出来，这样用户在使用Tab键穿越链接结构时一遇到它就知道可以使用：

```
a#skip-navigation:focus,
a#skip-navigation:active { left: 0; top: 0; }
```

有些人可能认为跳过链接不再是必需的，因为最新屏幕阅读器上的浏览器能更好地利用语义标记：它们提供了各种机制让用户能通过标题、列表和段落进行导航，同时还采用了WAI-ARIA规范，这个规范指导开发者用路标角色标识出页面中的主要版块，方便用户从一个版块跳到另一个。但是，跳过链接对“高级”用户来说仍然有用，因为他们偏爱使用键盘进行网站导航，而另一些用户则被限制只能使用键盘导航，比如那些使用前智能手机时代老式移动设备的用户。由于缺乏内置的方法来跳过重复的导航内容，这些情形中的用户必须通过滚屏才能到达正确位置，如果屏幕尺寸太小或者太窄就会让人难以使用。

3.3.5　使用title属性

给元素应用可选的title属性是一种将意义和帮助性说明加入标记的简单方法，还有一个额外的好处：默认情况下，大多数浏览器都会将title属性渲染成工具提示或者帮助文本块，当用户的鼠标悬停在相应的元素上时就会显示出来，如图3-4所示。一些屏幕阅读器也会朗读出title的值。



图3-4　鼠标悬停时，标签下方显示出原生的浏览器工具提示

在基础标记里，用title属性来标识元素（比如链接图像）是很有用的：

```
<label for="email" title="为了阻止垃圾邮件发送者，我们会发送一封确认邮件来确保这是个有效的Email地址">Email地址</label>
<input type="text" name="user" id="email" class="text" />
```

title属性可以应用到大多数元素上，而且它接受任何长度的文本短语，不过，最好让title的值保持简明扼要，因为过长的短语或者句子在作为工具提示显示时有可能会被截短，或者在屏幕阅读器阅读时让用户的注意力偏离主要内容。

我们还时常用这一功能帮助用户理解该如何使用某种增强版应用程序界面里的纯图标按钮。举个例子，如果某张数据表格里的行是可编辑的，用户可以删除它们，那么当用户将鼠标移到按钮上方时，我们就可以通过title属性以工具提示的形式提供操作说明文字。

在增强标记里，title的值可以用CSS和JavaScript转变成交互功能更丰富的自定义工具提示，甚至可以在鼠标移至页面某一区域时跟随鼠标移动。（第10章会介绍一种方法，将title的值增强成自定义样式的工具提示。）

注意 title属性可以用于link元素，包括引用外部样式表的链接。如果某个样式表在页面里被指定为preferred或alternate，就必须使用title属性给这个样式表引用指定一个唯一的名称：

```
<link rel="alternate stylesheet" href="styles.css" type="text/css" media="screen" title="enhanced">
```

3.4 建立一张 HTML 文档

运用上面介绍的指导原则，我们已经建立起一套稳固的系统，用于标记和组织我们的HTML，所以现在是时候将内容放入一张正确的HTML文档了。

HTML文档最基本的结构十分简单，每张格式良好的网页都始于一个html元素，它包含下列额外的元素。

- 一个head元素，它会包含title元素和所有meta标签、外部样式表和脚本引用（随后会介绍更多有关这些元素及其相关属性的信息）。
- 一个body元素，内含网页内容。

从结构上看，它就像是这样：

```
<html>
  <head></head>
  <body></body>
</html>
```

一张包含这种极精简结构的文档可以在浏览器上正常工作（技术上是这样）。但是，为了让它的外观和功能在各种浏览器上保持一致，还需要添加一些元素来标明此标记的语法并描述这张网页的概况。

3.4.1 DOCTYPE

一个有效的DOCTYPE声明是一个标签，应该出现在网站中每个页面里。它标明了网页使用的标记语法，并引用一个用来进行验证的特定标记规范。简单地说，就是下面这样。

- 它指示浏览器按照兼容标准的模式运作。没有它，浏览器就会运行于怪异模式（quirks mode），意思是浏览器会猜测用哪种方式才能最好地渲染标记和样式。
- 它提供了根据某个特定标记规范（比如XHTML 1.1）来验证网页的必要信息。

为了能正确工作，DOCTYPE必须是网页里列出的第一个标签（在它前面不能有其他标签，甚至连空格也不行），还必须遵循某种特定的语法。大多数DOCTYPE必须包含一个对文档类型定义（Document Type Definition，DTD。它定义了文档使用的标记语言类型和版本）的引用，以及一个用来进行验证的链接，该链接指向发布在W3C服务器上的相关DTD。

每种标记语言都会指定一个特有的DOCTYPE标签。举个例子，HTML4.0和XHTML 1.1的标签都是独一无二的，第一个引号里的值是DTD，第二个值则是DTD链接：

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN" "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
```

下面这个HTML5的DOCTYPE也是有效的，不过它缺少了DTD和链接：

```
<!DOCTYPE html>
```

为什么这个DOCTYPE会不一样？W3C想要消除标注特定HTML版本的必要，鼓励开发者直接按照Web标准编写代码。采用HTML5DOCTYPE的文档会按照HTML5规范进行验证，HTML5规范允许使用XHTML的语法（比如关闭所有标签），并指示浏览器用兼容标准的模式进行渲染。

为什么要验证代码

“无效”标记指的就是不完全遵照相应规范要求编写的标记。举个例子，用一个空格和斜杠关闭某个图像标签在XHTML 1.1里是有效的，但在HTML4.0里则是无效的：

```
<img src="tree.gif" alt="美国榆树" />
```

图像在有效的HTML4.0里是不关闭的：

```
<img src="tree.gif" alt="美国榆树">
```

在大多数情况下，无效标记仍然会被正确渲染。事实上，许多网站（包括那些由行业领导者创建的）所使用的并不都是有效标记（参见“The XHTML 100”：www.goer.org/Journal/2003/04/the_xhtml_100.html）。

既然无效标记也能正常工作，那么验证代码到底是为了什么？下面两个原因解释了为什么它仍然是我们开发实践中的一个核心部分。

- 代码验证经常会揭露出标记中的隐藏问题，这些问题本来要很久以后才会显露出来，而那时修复它们会困难得多。
- 一张验证过的网页遵循Web标准，因此是向前兼容的（validator.w3.org/docs/why.html）。

3.4.2　文档头

一张HTML网页的head部分预留给外部样式表和JavaScript文件的链接，以及一些关于内容的描述性信息，这些信息可能会帮助你的网页更好地配合搜索引擎工作。正确配置这些元素会帮助你的网页更好地工作，还能方便用户代理（包括标准浏览器和屏幕阅读器）进行解读。我们会讲解正确组织网页head会用到的最重要的元素。

1. 内容类型和字符编码

内容类型（或者叫MIME类型）告诉浏览器要准备处理什么样的标记，它可以通过在网页head里添加一个meta标签进行设置：

```
<!doctype html>
 <html>
   <head>
     <meta http-equiv="content-type" content="text/html" />
   </head>
   <body></body>
 </html>
```

还可以全局性设置网站的内容类型，方法是在服务器级别上将它定义为http-equiv头属性。举个例子，在Apache服务器上你可以指定一个.htaccess文件，内含用于整个网站的内容类型：

```
<Files abc.html>
 ForceType text/html; charset=UTF-8
</Files>
```

像内容类型一样，还可以指定字符编码的类型，也就是浏览器应该如何解释写在标记中的字符。举个例子，假如你的内容用的主要是俄文，就应该指定一个包含西里尔（Cyrillic）字母表的字符编码。网页中最常用的编码包括。

- UTF-8。它基于Unicode编码模式，向后兼容ASCII，而且覆盖的字符范围比ISO-8859-1更广，从西欧语言到东亚语言都有。
- ISO-8859-1。它基于ASCII，主要包括了西欧字符（http://en.wikipedia.org/wiki/ISO/IEC_8859-1）。如果文档的内容类型是text/html但没有指定编码，ISO-8859-1就会成为默认编码（www.w3.org/Protocols/rfc2616/rfc2616-sec3.html，参见3.7.1节）。

我们推荐使用UTF-8，因为它支持的语言范围比ISO-8859-1更广。

可以在服务器设置的同一个http-equiv头属性里指定字符编码，也可以在同一个meta标签里进行文档级别的设置：

```
<!doctype html>
<html>
  <head>
    <meta http-equiv="content-type" content="text/html; charset=utf-8" />
  </head>
  <body></body>
</html>
```

> **注意**　不包括在指定编码里的字符同样可以写入网页标记，形式可以是命名实体（named entity），比如用 代表不间断空格，也可以是数字字符引用（Numeric Character Reference，NCR），它用 代表不间断空格。
>
> 请访问下列网站，了解所有编码参考信息：
>
> www.webstandards.org/learn/articles/askw3c/dec2002
>
> http://en.wikipedia.org/wiki/Character_encoding

2. 网页标题

客户经常会问我们该如何改善他们网站在搜索结果列表里的位置。一个简单的办法就是确保所有网页都有一个独一无二的描述性标题：

```
<html>
  <head>
    <title>我们做过些什么：UI设计和开发客户案例 | 马萨诸塞州波士顿Filament集团公司</title>
  </head>
  <body></body>
</html>
```

每张网页都应该有一个独一无二的title元素来描述网页的内容和用途。如果你给网站里的每张网页都设置相同的样板标题（比如：Acme集团公司），那么不但会伤害到你的搜索引擎优化（Search Engine Optimization， SEO）排名，对访问者而言也不是非常有用。撰写标题时，请注意下面这些事实。

- 浏览器会将标题用作默认书签名以及显示在标签栏里，因此要让开头的一两个单词尽可能地展现出特征性。
- Google或Bing等搜索引擎将标题用于每个结果的链接中，并将其截短成大约60~70个字符的单行文字（不过一般都认为它们仍然会将整个标题纳入考虑范围）。要使用简洁的语言，并将最能具体描述网页的单词放在开头，这些做法会帮助标题在搜索列表里突显出来。
- 用户头脑中的合适网页标题用词可能和你（比如说制造商）想的不同。要确保使用以顾客为中心的语言，符合你目标客户的预期，而不是使用公司行话或者技术名词。举个例子，对一个产品页来说，“产品详情：Acme Solarfox 100B手提灯”就不如“太阳能手提式LED野营灯 - Solarfox 100B”有用。
- 如果你把你的名字（或公司名称）添加到每张网页的标题里，请确保将其列在最后，着重强调那些描述网页内容的文字。
- 最后，可以考虑使用地区性的关键词，帮助搜索引擎了解你所处的位置，例如“美国马萨诸塞州波士顿”。但同样也请把它们放在标题的最后。

标题还应该简明：不要仅仅为了提升网页访问量就往标题里塞入不必要的关键词。一些搜索引擎可能会检测到网页标题里充斥着大量重复或者多余的关键词，给它们的排名反而会更低，甚至整个网站都会在搜索结果页里清除。总而言之，一个准确、精心选择的网页标题（比如“太阳

能手提式LED野营灯 - Solarfox 100B - 美国马萨诸塞州波士顿Acme公司”）无论是对人还是搜索引擎都会提供更有用的信息。

3. 网页描述和关键词的META标签

关键词和网页描述的meta标签原本设计用于向搜索引擎提供网页内容的摘要。多年来，SEO营销者们是如此严重地滥用这些标签，以至于一些人猜测它们在搜索引擎排名或相关性上不再占据多少权重。不过，给网页添加一组精心选择的meta标签仍然有用，因为它们能给文档增加有价值的意义。

网页描述meta标签的内容有时会显示在搜索引擎结果页或其他环境里。因此，它应该组织成一段关于网页用途的简单描述。它应该用一两句话概括网页的内容或者功能，并使用能补充说明网页标题的文字，但这些文字要更具描述性和对话性。它也可以比标题更长一些，因为它并不像后者那样受具体字符计数的限制：

```
<meta name="description" content="Solarfox 100 太阳能手提式LED野营灯，颜色多样且完全防水，由Acme公司在美国制造。" />
```

相比之下，关键词meta标签则主要用来告知搜索引擎蜘蛛和其他自动化系统，该如何解释和评估网页里的关键信息：

```
<meta name="keywords" content="solarfox, 太阳能, 可充电, 手提灯, 绿色技术, 野营, acme, 美国制造, 波士顿, 马萨诸塞州, 新英格兰" />
```

要以逗号分隔列表的形式提供用户可能搜索的那些最重要的术语，包括相同主题的逻辑变体（即太阳能和可充电）。与标题类似，如果你将列表做得过长或者包含重复的词汇，网页就可能变得臃肿，理论上会伤害到它的排名，所以最好将列表控制在40个单词以内。

4. 语言

语言meta标签的作用是标明HTML网页使用的语种，并帮助用户代理解读内容：

```
<meta name="language" content="en-us" />
```

每种语言都有一个代码，有时还会附带一个后缀来区分特定的方言（例如用en-us来指代美式英语）。语言代码及其使用方式的相关语法请参见HTML规范里的概述（www.w3.org/TR/html401/struct/dirlang.html#langcodes）。

5. 机器人

机器人（robot）meta标签的作用是告诉搜索引擎蜘蛛该如何索引网页内容，以及是否可以跟踪网页标记里列出的那些链接。

默认情况下，蜘蛛会索引所有网页内容并跟踪一切链接，这就等同于用下面的标签明确指示搜索引擎这么做：

```
<meta name="robots" content="all" />
```

如果想告诉某个机器人不要索引内容，应将content属性设置为noindex。要避免蜘蛛跟踪链接则需要添加nofollow属性，就像这样：

```
<meta name="robots" content="noindex, nofollow" />
```

要注意的是，不是所有的蜘蛛都会遵守你的机器人标签，所以应该采取混淆（obscuring）或加密等预防措施来保护敏感内容，阻止那些扫描Email地址和敏感信息的垃圾邮件发送者和其他恶意软件。

6. 样式表和脚本引用

将样式规则和JavaScript放在外部文件里，然后通过head引用来组织和管理这些资产是一种最符合标准的方式，也是最有利于渐进增强的。

每张样式表都应该用一个link元素引用，加上绝对或相对路径，然后放在所有meta标签的后面：

```
<link rel="stylesheet" type="text/css" href="/styles/basic.css" />
```

默认情况下，样式表会以同样的方式应用于一切设备和格式（在设备或格式允许的范围内）。可以让它有针对性地影响在PC（screen）或移动设备（handheld）上显示的网页，以及打印（print）的网页，方法是指定适当的media属性：

```
<link rel="stylesheet" type="text/css" media="print" href="/styles/print.css" />
```

JavaScript文件引用（用script标签表示）原则上应该跟在所有样式表的后面。这样就确保了样式会先应用到页面，然后才是交互：

```
<script type="text/javascript" src="/scripts/ajax.js"></script>
```

7. 网站图标

建立HTML文档的画龙点睛之笔是引用一个网站图标（即favicon）图像。设置图标后，它会出现在浏览器的地址栏、标签和收藏夹菜单里，方便用户识别你的网页。图像的尺寸至少应该有16×16像素。虽然现代浏览器会接受任何图像格式的网站图标（包括JPG、GIF或PNG），但为了让它能出现在种类尽可能多的浏览器上，建议使用传统的图标（ICO）格式，它的扩展名是.ico。只需像这样引用它：

```
<link rel="shortcut icon" href="/favicon.ico" />
```

有许多网页工具可以将JPG、GIF或PNG图标转换成ICO格式。我们用过下面这些，效果不错：

- www.favicon.cc
- www.degraeve.com/favicon

提示　如果网站有许多苹果iPhone和iPod touch用户，还可以创建一个特殊图标，使它出现在设备的主屏上。图像必须是57×57像素大小、PNG格式，文件名为apple-touch-icon.png：

```
<link rel="apple-touch-icon" href="/apple-touch-icon.png" />
```

3.5　加入可访问性

简单地说，创建可访问的标记就是指编写简洁、组织良好和标识清晰的标记，并遵守一些简单设计规则，确保文本易读、链接的自解释性良好。按照这些标准编写的标记对屏幕阅读器或其

他辅助技术、搜索引擎和传统浏览器用户而言都有更高的可用性，因为它编入了额外的意义和帮助性的反馈，两者都有助于提高网站的整体可用性。这是一个所有人共赢的选择。

以下两组指导原则概括了你应该在代码中努力满足的具体标准：508条款（Section 508）和Web内容可访问性指南（Web Content Accessibility Guidelines，WCAG）。当你思考如何实现完整可访问网站时，要将两者都考虑在内，所以这里先总体介绍一下，然后再在后续章节里分析具体的可访问性案例。

注意　遵循此处概述的指导原则并不能保证你的网站一定是可访问的。唯一的确认方式是将全面的可访问性测试纳入到你的质量保证流程之中。

3.5.1　可访问性指导原则和法律标准

许多国内和国际标准描述了不同司法机构在法律上可以接受的可访问性措施，包括美国的“美国残疾人法案”（Americans with Disabilities Act，ADA）第508条款，以及欧盟的互动媒体指导原则。

这些规则和标准适用于从软件应用程序到电信产品等一系列信息技术。举个例子，美国ADA第508条款中包括了许多特别适用于Web开发的标准。

a 每个非文本元素都必须提供等价的替代文本（即通过`alt`属性提供，或在元素内容里提供）。
b 任何多媒体表达方式的等价替代物都必须与该表达方式保持同步。
c 网页的设计必须让所有附带颜色的信息在无颜色时同样可用，比如来自上下文或者标记。
d 文档必须组织成无需关联样式表仍然能阅读的形式。
e 必须为服务器端图像映射（image map）里的每个活动区域都提供重复的文字链接。
f 必须用客户端图像映射取代服务器端图像映射，除非该区域无法通过可用的几何形状进行定义。
g 数据表格必须标明行标题和列标题。
h 如果数据表格里的行或列标题有两个或更多逻辑层级，则必须使用标记关联数据单元格和标题单元格。
i 框架必须带有文字标题以帮助用户识别框架和导航。
j 网页的设计必须避免让屏幕以高于2 Hz且低于55 Hz的频率闪烁。
k 如果某一网站无法用其他方式遵守此部分里的条款，则必须提供拥有等价信息和功能的纯文本网页。纯文本网页里的内容必须与主网页同步更新。
l 网页使用脚本语言来显示内容或创建界面元素时，该脚本提供的内容必须用能被辅助技术正常读取的文本进行标识。
m 一张网页要求客户系统中存在某个小应用程序（applet）、插件或其他应用程序来解释网页内容时，它必须提供一个链接，指向符合§1194.21(a)到(l)标准的插件或小应用程序。

n 如果电子表单的设计要求在线完成，则该表单必须让使用辅助技术的人能够访问到完成和提交这一表单所需的信息、字段元素和功能，包括所有的指引和提示。

o 必须提供一种方法，让用户可以跳过重复性的导航链接。

p 如果要求限时应答，则必须警示用户，并给予其足够的时间来表明自己需要更多时间。

我们建议我们的客户，特别是那些为政府机构或接受政府资助的各类人群（比如学校）提供服务的客户，理解他们在这些标准下所承担的义务。而且总的来说，我们建议尽可能多地运用它们，将其作为良好的编程实践。

要了解508条款中的标准及如何运用它们的更多信息，请参见508条款的网站：www.section-508.gov。

3.5.2　Web内容可访问性指南

Web内容可访问性指南（WCAG，读音为wh-kag）当前的版本是2.0，它由WCAG工作小组创建，经万维网联盟审查和批准。万维网联盟本身是制定并维护标记和CSS等基础规范的标准组织。

在宏观层面上，WCAG提倡编码时遵循以下四条总体原则：

- 让万物可感知；
- 让万物可操控；
- 让万物可理解；
- 让万物很健壮。

在具体实践中，WCAG规定了以下一些常识性规则来实现这些总体目标。

- 为任何非文本的内容提供替代文本，这样它就能被转换成人们需要的其他形式，比如大字体版、布莱叶（Braille）盲文、语音、符号或者更简单的语言。
- 为基于时间的媒体提供替代物。（基于时间的媒体包括预先录制或现场版的视频和音频。）
- 创建可以用不同方式呈现（比如更简单的布局）而不丢失信息或结构的内容。
- 让用户易于看见和听到内容，包括将前景和背景区分开来。
- 让键盘可以实现所有功能。
- 给用户提供足够多的时间来阅读和使用内容。
- 如果已知某种方式会诱发癫痫，则不要将其用到内容设计中。
- 提供一些方法帮助用户导航、找到内容和判断所处位置。
- 让文本内容易于阅读和理解。
- 让网页以可预期的方式出现和运作。
- 帮助用户避免和纠正错误。
- 与当前和未来的用户代理（包括辅助技术）保持最大程度的兼容性。

阅读完整列表（以及详细解释和使用指导原则）请访问WCAG的网站：www.w3.org/TR/WCAG。

我们已经（希望如此）介绍了如何建立一张结构良好、能够驱动基本体验并为增强体验打下基础的语义化HTML网页。接下来讨论如何使用CSS对渐进增强来说最有效。

第 4 章

有效应用样式

层叠样式表（Cascading Style Sheets，CSS）定义了网页的视觉样式，并让表现和内容能够清楚分开。一系列的高级字体技巧、视觉特效和复杂布局正是因为CSS才成为可能。正确应用这些样式会极大提高网站的美感和可用性。

渐进增强和现代编程实践的基本原则之一就是将表现从标记中分离出来，方法是将所有的样式规则组合到一个或多个外部样式表中。虽然将所有样式都写入外部样式表是渐进增强里必不可少的一种最佳实践，但这种做法本身却并不能保证为所有用户提供令人满意的体验。每一种浏览器都会尝试渲染网页里引用的所有样式表，无论它是否能正确支持里面指定的CSS属性。如果在能胜任的设备里渲染，样式表可以让网页变得更方便阅读，或让工作流更易于执行，但如果所渲染的浏览器只能部分支持CSS，它就可能使体验变得令人困惑和无法使用。

使用渐进增强方式进行开发时，我们在分离表现这个基本原则上更进了一步：将样式分离成"安全"样式，即可以随基础标记输出给所有浏览器（包括那些只是部分支持CSS的浏览器），然后用EnhanceJS测试套件对浏览器进行测试，判断它是否有正确渲染的能力，只有测试通过后我们才会加入更复杂的样式增强信息和CSS技巧。

这一章会回顾在渐进增强里应用CSS时经常采用的最佳实践，包括介绍如何将样式分入基本和增强样式表，并重点介绍一些技巧来改进可访问性、可用性和样式的性能。

4.1 将样式应用到网页

许多CSS最佳实践能全局性地适用于基本体验和增强体验。这些至关重要的做法包括要组织外部样式表里的样式，使它们能根据浏览器的能力合乎逻辑地区别应用；要正确引用它们；要在合适的环境里做出明智的指向；要使用有意义的命名惯例，根据用途或功能而不是它们的视觉特性来描述样式。

4.1.1 将样式保存在外部样式表里

保持标记和样式清楚分离的第一步，是在一个外部样式表里引用所有样式。这种将样式从标记中分离出来的最佳实践不但被开发者广泛采用，而且很重要，因为：

- 它让网页可以呈现多种样式，而且容易更换，因为它没有"烙印上"某种特定的外观；

- 它将所有的样式规则保存在同一个地方，从而简化了维护工作；
- 它通过公用类和ID将共享样式集中到一起，而不是在标记里多次重复它们，这样能最大程度减小代码体积。

这种方法与常用（但经常会出问题）的实践做法，即内联写入样式存在着根本的不同：后者可以用style属性给标记里的具体元素指定样式（比如style="padding: 5px"），或者写在网页的某个样式块里。内联样式会带来许多麻烦。

- 它们无法被网页或网站里的多个元素重复利用，这就意味着内联样式必须在每一个需要它的地方都复制一次。
- 它们难以维护和升级，因为它们内嵌于HTML内容之中，而不是存储在一个单独的外部位置上。从长期看，这会成为让人头疼的维护问题，特别是当设计出现任何显著变化时。
- 它们不受外部CSS所引用的样式规则控制。即使全局样式是在外部维护的，标记里的内联样式规则实例仍然会覆盖外部的全局样式。
- 随标记输出的样式会被浏览器渲染，哪怕它们并不没有完全得到支持。不是所有浏览器都能以相同的方式处理CSS，它们甚至不一定完全支持所有的CSS2属性，因此在某些情形下内联样式可能会导致内容无法使用。

即使有些时候某种样式在整个网站里只使用了一两次，我们也建议将它列入外部样式表，和网站的其他所有样式规则放在一起。

可以使用内联样式的场合：几种例外情况

在高级UI设计的几处非常特殊的场合里，我们发现有必要使用内联样式：动态放置屏幕上的元素，动画式修改元素样式属性，以及创建基于CSS的图表。

举个例子，如果你需要在屏幕上居中显示一个模式对话框（modal dialog），或者重复设置某个元素的透明度来创造淡出渐变效果，那么为每一个可能的位置坐标组或动画运动增量创建一个类是不切实际的。在这些案例中，可行的做法是用JavaScript添加一个style，然后即时修改CSS属性。我们还使用内联尺寸属性创建纯粹基于CSS的图表，方法是给图表里每一个数值条的对应元素设置百分比形式的宽度和高度样式。

话虽这么说，我们还是建议尽量避免使用这些内联样式规则，只要有可能，请使用外部引用的CSS来制作视觉表现。

4.1.2　链接到外部样式表

开始为某个项目编写代码时，初始工作之一就是创建一张样式表，用于包含基础标记所需的全部样式规则，并在网页的head里引用它：

```
<link type="text/css" rel="stylesheet" href="basic.css" />
```

有效的样式表链接包括以下这些属性。

- type指定内容类型，供服务器解释（对样式表来说，它的值永远应该是text/css）。
- rel声明被链接资源与HTML文档之间的关系（stylesheet）。

▶ href包含一个CSS文件的相对或绝对路径 。

根据HTML4规范，link元素只有列入文档head内才有效，但是把它放在那里还有一个更重要的原因：任何列入head的资源都会先于body里的元素加载。换句话说，将link元素放在head里能让样式先就位，然后网页内容才会显现，因此它们一出现就已经带着合适的样式了。在网页的其他位置引用样式表增加了网页无样式加载的可能性。随着样式的加载，用户会看到设计出现一步或多步的转变，而这往往不是最佳的用户体验。

还可以在一张网页里引用多个样式表。这样做时，请注意页面里引用的每个样式表都会对服务器构成一次单独的HTTP请求，而且浏览器在显示页面内容之前会先加载HTML网页head块里引用的所有样式表。因此，我们建议把CSS组织到一起，使文件的数量尽可能少些。

通过@import引用弊大于利

某些开发者偏爱使用@import标识来引用外部样式表，将它放在style标签或者别的样式表里，与其他的CSS选择器（selector）并列。这个标识经常被用于组织多个样式表以及管理它们之间的依赖关系。举个例子，只需在网页引用的主样式表里加几条@import规则就能组织起屏幕表现所需的全部样式：

```
@import url(screen-global.css);
@import url(screen-article-formatting.css);
@import url(screen-admin-tools.css);
```

虽然用这种方法组织CSS文件看似很有效率,但是@import实际上可能会减慢网页载入过程。每一条@import引用都需要浏览器单独发起一次完整的HTTP请求到服务器。另外，@import请求是异步进行的，就是说浏览器不会等待这些请求加载完成后才开始渲染网页内容。这种方式更有可能导致用户在网页开始载入时看到朴素无样式的HTML，因为样式还没有完全加载到位。

因此，我们建议引用样式表时绕过这种方法，用link标签代替，并尽可能减少文件数量。

针对特定媒介类型

默认情况下，用link元素引用的样式表会应用到所有媒介形式上，包括标准计算机屏幕、移动设备和打印机。

要标注某个样式表，让它只用于某种特定形式，应使用link元素的media属性来设置一个或多个媒介类型（多个值之间用逗号分隔）。举个例子，这个样式表只会将格式信息应用到打印输出上：

```
<link type="text/css" rel="stylesheet" href="basic.css" media="print" />
```

还可以给一个样式表里的各个CSS块指定媒介类型。要在一张用于多种媒介形式的样式表内，对某个特定类型指定一些例外的样式，可以这么写：

```
@media print {
  h1 { font-size: 16pt; }
```

```
  h2 { font-size: 14pt; }
  h3 { font-size: 12pt; }
  #navigation, #advertisement { display: none; }
}
```

HTML规范列举了一些可供选择的媒介类型，其中的三种适用于网站和应用程序。

- screen在标准计算机屏幕上的浏览器里渲染所有关联样式。
- print将格式信息应用到打印机输出上，其布局通常都设置成点（point）和英寸，尺寸则经常根据纸张大小而定。
- handheld将样式应用到手持设备上，这些设备拥有低速的网络连接、较小的屏幕和有限的图形处理能力。

很遗憾，其中的handheld媒介类型是最不可靠的。写作本书时，很少有移动浏览器会真正遵从这一媒介类型，并以此解析样式。要确保向移动设备输出合适的样式，最好的办法是依靠测试驱动的渐进增强方式进行开发，使那些网速和性能较低的设备能够获得可用的体验。

4.1.3　使用有意义的命名惯例

在为类和ID命名时，我们建议使用能够描述内容用途或内容层级角色（而不是其屏幕呈现方式）的名称。可以把类和ID看做语义标记的延伸——名称是否准确归类和标明了所包含的内容？

选择有意义的命名惯例对渐进增强开发来说是一种最佳实践，原因有两个。

- 带有类或ID标记的元素可以在基本和增强体验里应用不同的样式。举个例子，基本网站里的一栏文本在增强版里可以被转换成模式对话框。根据内容的用途为其命名会生成一个符合逻辑描述的挂钩，通过它既能应用安全和增强样式，也能应用JavaScript交互。
- 即使在同一个开发周期里，视觉设计也可能发生变化。举个例子，某个左侧导航条可能会被移动到页面的顶部或者右侧，这时如果它的ID名是left-nav就讲不通了。更好的替代方式是使用能够描述该导航用途的名称，比如primary-nav或者secondary-nav。

如果CSS可能由多位开发者进行维护或扩展，那么这一点就尤为重要。作为一家专业服务公司，我们编写的代码中有90%会移交给客户，或者以开放源代码的形式提供给公众。因此，我们使用的命名惯例应该能让那些没有参与初始构建流程的开发者理解。要让名称简单和具有目的性，避免使用那些只有开发团队成员才知道的隐晦缩写词或者行业术语。这样做，代码在项目的整个生命周期里就会变得更加健壮。

4.2　为基本和增强体验添加样式

在测试驱动的PE（渐进增强）方法里引用外部CSS时，还需要额外考虑一些因素。因为借助能力测试框架，我们能为同一张网页提供基本和增强两种表现方式，所以需要确定CSS里的哪些部分应该发送给所有人，哪些部分应该为使用有能力浏览器的用户保留。

利用这个测试框架，我们能够非常精确地划分CSS，然后分别输出给全部浏览器（包括那些未通过部分能力测试套件的）和那些能通过所有测试的浏览器。为了让接下来的例子更明确，我

们假设两种体验用到的样式会分别储存在名为basic.css和enhanced.css的样式表中。basic.css里包含的任何样式都会在一开始输出的网页源代码里引用，如果浏览器通过了能力测试，我们则同样会将enhanced.css加载到网页中。（第6章会详细介绍使用该框架引用这些CSS文件的机制。）

4.2.1　基本体验里的安全样式

大多数浏览器都有一个默认样式表，这些默认设置能很好地表达内容层级，所以我们一般会优先使用浏览器的样式表，并尽量少用自定义样式覆盖它。我们在运用测试驱动的PE方法时发现了一些“安全”的CSS属性，它们可以用在基本体验里，所以通常会在basic.css样式表里设置它们。这些属性在各种浏览器里的渲染效果比较接近，无论浏览器是新是旧，还是对CSS支持程度不一都是如此。虽然有些浏览器不会遵守这些属性，但即使识别不出它们，也不会弄乱网站。

- ▶ **文本格式**：要设置网站的默认字体，你可以在body标签里“堆出”一列字体，并把优先使用的放在最前面（`font-family: "Segoe UI", Helvetica, Arial, san-serif`）。我们建议这堆字体里包含众多系统里通常都安装了的字体。其他的文本格式，比如粗体（`font-weight: bold`）、斜体（`font-style: italic`）、删除线（`text-decoration: line-through`）、转换成大写（`text-transform: uppercase`）、对齐（`text-align: center`）、行距（`line-height:1.2`）和字距（`letter-spacing: 0.3em`）在绝大多数浏览器里都能得到比较好的支持。（我们建议*不要*在基本体验里设置字体大小，或改变`text-decoration`属性来修改默认的链接下划线。）
- ▶ **文本颜色和背景**：总体而言，文本颜色（`color: red`）、背景颜色（`background-color:#224466`）和背景图像（`background-image: url(images/texture.png)`）都可以安全地用在基本体验里，前提是文本在未应用背景图像或背景色时仍然清晰可读。请注意，有些移动浏览器支持文本颜色但不支持背景颜色，因此反白文本（例如深色背景中的浅色文本）可能会不可见。另外，需要重点注意的是，许多移动浏览器总是会将链接的文字渲染成蓝色，所以哪怕你指定了有效的链接颜色，最终的链接文字可能还是蓝色。（如果背景是深蓝色，链接也许会完全不可见。）我们建议你遵循一条经验法则：将反白文本留到增强体验里使用。
- ▶ **内边距和外边距**：内边距（`padding:1em`）和外边距（`margin-top:1.5em`）能够将元素分隔开，增强网页在基本体验里的可读性，但是使用次数应该严格加以控制。永远不要在基本样式表里设置浮动、定位、堆叠或溢出属性。
- ▶ **边框**：边框（`border:1px solid black`）可以适量使用，它能将用户的注意力集中到各个内容群组上，大大改善基本体验。

这些样式属性结合起来一般足以给网站打上简单的品牌标记，让它变得独特和可识别，并增强网站的用户友好性。

让basic.css保持简单干净还有另外一个好处：它可以成为基本屏幕体验和打印端共用的样式表。为此，别忘了要添加打印端专用的`@media`块到basic.css中，隐藏增强体验里那些与打印无关的元素（导航、广告等）：

```
@media print {
  h1 { font-size: 16pt; }
  h2 { font-size: 14pt; }
  h3 { font-size: 12pt; }
  #navigation, #advertisement { display: none; }
}
```

4.2.2　为增强体验添加样式

通过使用能力测试，将增强信息只发送给能够理解它们的浏览器，我们就可以放心地应用高级CSS布局技术（比如浮动、定位等），而不必担心它们是否会造成可用性问题。这一实践还允许在编写增强版CSS时假定JavaScript已存在并得到良好支持。

编写增强版CSS时遵循许多最佳实践，建议你考虑一下。

- **把工作交给CSS**。如果某个网页组件具备多种动态视觉状态（比如默认、高亮或禁用状态），对用户的输入或操作做出响应，则应该为每种状态编写一个类，用JavaScript切换它的开启和关闭。这样CSS就承担了主要工作，从而避免出现用JavaScript添加不必要的内联样式这种情况。
- **利用好样式表的层叠**。如果你在编写基本CSS时注意适度，就能够方便地用增强版CSS扩展它，而不必去抵消那些已经声明过的样式。这就是我们提倡在基本体验里尽量少用CSS的原因。
- **少用重置样式**。如果你的增强版CSS规则需要标准化或者抵消浏览器的默认样式，建议把它们放在enhanced.css的顶部（并且完全避免在basic.css里使用它们）。强烈建议尽可能少重置样式，而应在浏览器提供的样式基础之上进行构建。这点对表单元素而言尤其重要，因为它们自带的某些特定默认样式是用户识别的依据，所以只要有可能最好别去碰它们。
- **如果可行，就采用前沿的CSS**。许多较新的CSS3功能提供了丰富的视觉样式（例如圆角、阴影、可变不透明度和动画渐变），并逐步得到最新浏览器的支持。它们能降低代码的臃肿程度，还提供了强大的选择器，将样式规则关联到标记结构上。举个例子，如果使用CSS3的`border-radius`将某个元素的边角圆化，就无需再借助多个HTML元素来存放方块四角所需的背景图像。虽然这些功能有很多在IE 6等旧式浏览器里无法工作，但是在增强体验里使用它们是安全的，因为不能识别它们的浏览器会忽略它们。
- **用图像设置字体要留神**。一些设计可能会用到没有在计算机和设备上普遍安装的自定义字体。我们从来不推荐用图像来设置字体（哪怕同时还应用了可访问技术，比如`alt`属性）。有许多自定义字体技巧能够确保网页内容对所有用户（包括那些使用屏幕阅读器的用户）都可用，而且还可以将网页内容用多种语言提供。自定义字体可以在Canvas或Flash中进行渲染，从而从视觉上替换部分网页文本，而且许多浏览器还支持用CSS的`@font-face`通过URL引用自定义字体。

4.3 可访问性的考虑要点

给网页应用样式时，需要为所有用户（包括那些残疾用户）考虑相当多的可用性和可访问性事项。

W3C提供了一份名为“Web内容可访问性指南 2.0”（WCAG）的详尽建议，它涵盖了一系列可访问性最佳实践，包括选择有意义的标记，视觉设计考虑要点以及使用JavaScript改进可访问性。其中有一些我们认为非常重要，值得强调一下。

- **文字大小与缩放**：设计网页时要将文字大小设置得足够大，使一般用户能够轻松阅读。用可变单位（例如em和%）或关键词（例如`small`、`medium`和`large`）编写的CSS能方便用户根据个人偏好调整文字大小。虽然大多数现代浏览器能够在任何CSS单位下对网页内容进行缩放，但是IE 6和之前版本只有使用可变单位时，文本缩放才能生效。
- **对比度**：要确保广大受众能够舒服地阅读文本内容，WCAG推荐将大尺寸文字的前景/背景对比度设置成不小于3：1，小尺寸文字则是5：1。（作为参照，黑色文字在白色背景里是21：1，而hex#666这样的中灰色在白色背景里则是5.74：1。）有许多工具可以用来测试具体的对比度，确保它处于可接受的水平。
- **颜色适应性**：许多界面通常会使用基于颜色的视觉提示，例如链接的彩色文字，用红/绿色指示性能，以及高亮警示信息。有6%~9%的人患有不同程度的色盲，其中最常见的是红绿色盲。因此，任何视觉指示都应该既有颜色值的差别，又有辅助的视觉指示信息（比如明显不同的形状或图案），让色盲人群能够使用。
- **明确的链接功能可见性**：用户会预期位于页面某些特定位置（顶部、沿着左侧栏和底部）的文字无论样式如何都是链接。对于页面主体内的文字链接而言，重要的是要让它们的样式与周围文字存在足够大的区别，并可以考虑始终给它们加上下划线，或者至少在鼠标悬停时添加，以达到最佳的区分度。
- **备用背景色**：给文本添加背景图像时，请确保在图像下方补充设置一种背景色作为备用，以防图像未能加载或者用户禁用了图像。（在这种情况下，也应注意上面介绍的对比度考虑要点。）

另外，我们自己还有许多可访问性的最佳实践原则，在考虑整体页面布局和结构时一直会将它们牢记于心。

- **考虑屏幕分辨率**：设计宽度固定的网页时，建议将设计宽度限制在960像素这一常用屏幕分辨率之内，以确保绝大多数用户无需横向滚动。
- **呈现舒适的列宽和行宽**：大约80个字符（9~10个单词）宽的文本列阅读起来最舒适。对那些文字量超大的网站而言，我们会考虑将列宽设得更窄，以提高浏览和阅读的舒适度。
- **可能的话，使用浏览器原生的焦点样式**：大多数浏览器将焦点处理成点状或发光的轮廓线，让用户能容易地使用他们的键盘在网页里导航。虽然这种轮廓线处理方式有时会和

预想的用户界面设计冲突，但我们建议你不要禁用它。如果你这么做了，请确保创建出另一种在获得焦点时出现的样式（比如一个边框或者背景色发生改变），以向那些不用鼠标进行页面导航的用户提供明确的反馈。

- **遵循既有惯例**：用户经过网上无处不在的训练，已学会在特定的位置找到常用功能：例如主要导航区在页面的顶部或左侧；账户工具、购物车反馈和退出链接在右上方；以及用户帮助链接在页面底部。将元素放置在可预测和统一的位置能帮助用户高效地找到他们所需的内容。
- **安全地隐藏和显示内容**：每次使用CSS对视觉用户隐藏文字时，请确保用可访问的方式进行。使用`display: none`或`visibility: hidden`可能会让文字同时对屏幕阅读器用户隐藏起来，这就意味着他们将无法访问该内容。一种可访问的隐藏解决方案是将元素绝对定位到页面之外，离开视觉用户的视野，方法是设置一个很大的负数`left`值（但不要设置一个`top`值，因为这可能会降低可访问性）。另一种解决方案是给元素设置一个很大的负数文本缩进值和一个隐藏的`overflow`，将文本移出视野。
- **尽量避免网页内部的滚动区域**：`overflow`属性的作用是指定内容溢出容器边界时应该如何呈现（可见、被截短或者可滚动）。iPhone等移动浏览器支持滚动式溢出，但不会显示滚动条，而且需要两根手指才能滚动这些区域，这导致滚动区域难以使用。另一些则完全不支持滚动。

4.4　应对 bug 和浏览器差异

编写能在各种浏览器中表现一致的CSS，通常要比想象中难。兼容性补丁（Hack）是开发者用来面向或排除某种特定浏览器的“创造性”语法，不到万不得已不要使用它们，而且即使用也要有选择地运用，因为它们难以维护，而且在遵循标准的新版浏览器发布后可能会导致渲染问题。幸好，如果你使用CSS的方式正确，兼容性补丁通常可以避免。对那些无法避免的有限情形而言，我们有一些技巧向你推荐。

4.4.1　条件注释

Internet Explorer因其古怪的CSS处理方式在网页设计师中声名狼藉，但考虑到它在浏览器市场里的份额，即使是微小的渲染差别也是难以忽视的。如果你的页面布局无论如何就是不能在某些版本的Internet Explorer里正确渲染，对于这样的情况不妨使用条件注释。它是Internet Explorer的一项私有功能，用于指定哪些HTML标记只有IE才能看到。（因为它们使用标准的HTML注释语法，所以其他浏览器会忽略它们。）

当有必要为IE变通使用CSS时，可以将那些样式规则单独列在一张样式表里，然后在网页上的条件注释内部引用它：

```
<!--[if IE]>
  <link rel="stylesheet" type="text/css" href="ie_fixes.css" />
<![endif]-->
```

条件注释可针对特定版本的IE。我们总是建议用lt指定一个版本，或是某个特定版本之前的所有版本，这样更符合标准的新版IE就看不到它们了。举个例子，下面这个条件注释只能被IE 7之前发布的版本读取（就是说，早于IE 7）：

```
<!--[if lt 7]>
  <link rel="stylesheet" type="text/css" href="ie_fixes.css" />
<![endif]-->
```

在IE专用的样式表里编写CSS时，我们建议做到以下几点。

- **只针对例外情况**：IE专用样式表里包含的变通代码数量越少越好，用来层叠替换那些输出给所有浏览器的样式。
- **少使用滤镜**：许多版本的Internet Explorer包含了被称为滤镜的私有CSS属性，它们能实现阴影和模糊等高级效果，还能让IE对半透明PNG的渲染得到修复（见接下来的补充内容）。很遗憾，IE滤镜不是合法的CSS，还可能会拖慢网页性能。如果必须要用的话，请谨慎使用。

IE 5和IE 6里的PNG透明度

IE 7之前的版本并不原生支持PNG 24图像中的alpha透明度。因此，PNG图像的半透明区域在IE 5和IE 6里会表现为灰色或者蓝色。幸好，如果你把图像改为PNG 8保存，就可以在这些浏览器里实现更为优雅的备用显示模式，可以使用Adobe Fireworks或者网上一些基于命令行的免费脚本做到这一点。在这些浏览器里观察PNG 8图像时，它的所有半透明通道会转为全透明，在用户界面里就远远不像之前那样碍眼了。

4.4.2　常见问题和变通方法

我们曾经为种类繁多的浏览器创建能够正常工作的CSS布局，在这些经历中，我们注意到有一些问题似乎在每个项目里都会出现。通常会使用以下一些变通方法来解决它们。

1. 清除和围住浮动

当CSS浮动应用到某个元素后，其父元素（parent element）就不再能确定它的高度了。这一布局行为有时会与预期布局渲染方式相冲突，因此需要寻找一种变通方法，让父元素围住它的浮动子元素。这个问题最简单的解决方法是让父元素浮动起来，或者将它的overflow属性设置成auto，但是这种方法并非在所有情形下都适用。

注意　要了解更多细节，请参阅Eric Meyer的“Containing Floats”：http://complexspiral.com/publications/containing-floats。

还有一种方法，可以应用一种名为“清除补丁”（clearfix）的变通代码。清除补丁这一技巧的工作原理是将一个元素插入最后浮动的子元素之后，并用CSS赋予它block和clear属性。因为

这个新元素自身是不浮动的，所以它能让父元素确定它的位置并围住它。

清除补丁用到的CSS非常巧妙。它使用:after这个伪元素选择器（pseudo-element selector）创建一个元素（在这个案例里是单个字符）并添加样式，不需要使用JavaScript。只需将下列样式规则添加到增强样式表，并将class="clearfix"应用到某个元素，强制让它围住所有浮动的子元素：

```
.clearfix:after {
  content: " ";
  display: block;
  height: 0;
  clear: both;
  visibility: hidden;
}
```

这里的:after选择器并没有得到Internet Explorer 7和之前版本的支持，因此有必要添加另一种选择器，确保清除补丁技巧能够正常工作。zoom: 1这个私有属性会触发Internet Explorer里的hasLayout（本章后面会介绍），使该元素围住它的浮动子元素。请记住，好的做法是将IE专用的样式存放在单独的样式表中，然后使用条件注释引用该样式表：

```
/* Internet Explorer专用 */
.clearfix {
  zoom: 1;
}
```

还有一种不那么碍眼的替代方式可以用来在整个标记里指派clearfix类。可以在样式表里列出所有需要清除补丁样式规则的元素，从而指定该规则的作用范围：

```
#header:after,
div#primary-navigation:after,
#primary-content:after,
#footer:after {
  content: " ";
  display: block;
  height: 0;
  clear: both;
  visibility: hidden;
}
/* Internet Explorer专用 */
#header,
div#primary-navigation,
#primary-content,
#footer {
  zoom: 1;
}
```

这一改变能够从标记里去除那些视觉变通所需要的类，可以减少网页体积。

致　　谢

用清除补丁这种方法来清除浮动最初是在这篇文章里介绍的：“How To Clear Floats Without Structural Markup”。这篇文章出自*Position is Everything*（www.positioniseverything.net/easyc-learing.html）。

2. 处理z-index问题

在现代网站里，大多数用户界面组件不仅能够相互并列摆放，而且还会堆叠在其他元素的前面或后面。在CSS里，堆叠是用z-index属性实现的。

z-index属性可以应用到任何具有relative、absolute或fixed定位（由position属性赋值）的元素上，它的值是一个数字，代表它在网页其他元素中的垂直堆叠位置。z-index值较高的元素会出现在z-index值较低（或不存在）元素的前面。整个文档存在着一种堆叠的上下文环境（元素相对于它们在网页上的同级元素进行堆叠），而且每个应用了z-index属性的容器元素都会为子元素创建一套新的堆叠上下文环境，意思是子元素的z-index值是相对其他子元素（而非网页的其余部分）而言的。

Internet Explorer 7和之前的版本错误地解释了这一堆叠模型：它们为所有位置确定的元素都创建了一套新的堆叠上下文环境，甚至包括那些未指定z-index的元素。这可能导致堆叠元素的摆放顺序变得难以预测。举个例子，一个相对位置确定的fieldset元素内部绝对位置确定的工具提示，可能会出现在网页里其他确定了位置的元素背后，因为IE将它们视为堆叠元素，而它们其实应该出现在文档流里。

要为确定了绝对位置的元素解决这个问题，我们使用JavaScript将标记（在这个案例里是工具提示）附加到body元素末尾的前面。这么做确保了工具提示会处于文档的（而不是某个特定元素的）堆叠序列中，如果指定了一个足够高的z-index值，它就会出现在所有其他网页元素的前面。总而言之，我们推荐在动态添加一切覆盖内容时使用这种方法，因为这样就无需再猜测应该将某个元素插入到哪一层，才能正确叠加到所有其他网页内容的上方。

Internet Explorer的另一个z-index问题是，select、iframe、object（用于Adobe Flash等插件）等“窗口化”（windowed）元素以及任何有滚动条的元素都倾向于堆叠在其他所有元素前面，不管源代码顺序或者z-index值如何都是如此。可惜，这个问题的变通方法不太美观：我们使用JavaScript和CSS直接将一个隐藏的空白iframe元素，插入到想出现在网页其余部分前面的内容之后，让它的尺寸大到足够成为一堵墙，挡住窗口化内容，不让它们透过来。

提示 要以自动化的方式使用这个补丁，我们推荐使用jQuery的插件bgiframe（http://docs.jquery.com/Plugins/bgiframe）。

3. 修复Internet Explorer里的hasLayout问题

在Internet Explorer里测试CSS布局时，你必定会遇到看似随机的渲染bug：元素只有部分可见，行为不遵守样式规则，滚动时出现闪烁，或者在意料之外的网页位置上显示出文字碎片。

这些类型的bug通常是由IE渲染引擎自身的问题所造成的，常见原因是某个元素缺少被微软公司称为hasLayout的一种特性。hasLayout确切含义的细节还不太为人所知。总体来说，块级元素在IE的渲染引擎里被当做是具有hasLayout，但有时引擎会忘记将hasLayout应用到这些元素上。

有一些CSS属性可以应用到元素上触发hasLayout。

- position: relative;这个属性通常是最安全的hasLayout变通方法，我们一般会首先尝试它。
- zoom: 1;这个CSS属性是私有且无效的，但没有多大危害，它赋予某个元素100%的页面缩放级别，以默认的样子渲染它。
- height: 1%;或者就这个问题而言任何height设置都行，它通常能触发hasLayout。

要确保对标准支持更完善的未来版本浏览器看不到这些样式，你可以考虑将hasLayout样式补丁放在IE专用的样式表里，这些样式表只针对需要这一补丁的特定版本Internet Explorer。

我们介绍了如何有效地编写和组织你的样式，使它们能清楚地针对基本和增强体验。接下来讨论如何通过无缝插入JavaScript来转换基础标记，从而增强页面并添加交互性。

第 5 章

编写增强和交互脚本

5

从最开始的地位低下，经常被误认为是一种古怪的“玩具”语言，到如今成为世界上使用最为广泛的编程语言之一，JavaScript在短短的时间内经历了一段漫长的旅程。今天的浏览器构建在强大的处理引擎基础之上，这样我们就能用JavaScript创建媲美桌面端程序的丰富体验，而且执行速度和能力正在快速提升。

通过使用测试驱动的渐进增强方法，我们能以一种更有针对性的方式应用JavaScript，不仅能有力增强基础标记，创建用户想要的高交互性网络应用程序，还能确保照顾到了缺乏JavaScript支持的浏览器上的可访问性。

这一章会讨论如何无缝应用JavaScript来扩展一张网页的功能，同时维持甚至增强它的的可访问性。我们会从4个基本方面考虑JavaScript的开发：

- 应用JavaScript到HTML；
- 理解JavaScript在基本体验里的位置；
- 编写增强体验脚本；
- 保持和增强可用性与可访问性。

在这一过程中，我们会详细介绍无缝组织JavaScript的方法，并尽可能将它编写得通用些，这样就可以在不同环境里重复使用。

5.1 如何正确引用 JavaScript

将JavaScript驱动的行为与HTML标记分离，对实现渐进增强而言是一种至关重要的最佳实践。清楚的分离能让脚本增强信息针对有能力的浏览器。用PE方法编写代码时应该无缝应用JavaScript，方法是将它放进某个外部文件里，然后在网页的head里引用它。

5.1.1 避免内联JavaScript

经常能见到内联的JavaScript代码块甚至事件处理程序（比如onclick或者onmouseover等标记）添加到HTML标记里。内联JavaScript可能看上去像是一种添加交互性的简便方法，但是一些重要的理由证明使用这种方法不是一个好主意。

- 它增加了代码体积。事件被内联使用时，它们必须重复出现在标记里的每一个单独元素

上，这让网页的文件大小变得臃肿，拖慢了加载时间。

- 它增加了维护的复杂性。维护内联脚本需要你细致地注明各个事件实例分别写在标记里的哪个位置，然后辛苦地寻找并更新每一个直接将JavaScript内嵌到HTML文档内容里的实例。如果你忽略了某些单独的实例，它们就可能导致错误。
- 它在所有支持JavaScript的浏览器里都应用了行为，包括那些支持程度不佳的浏览器。如果某些脚本技巧不能正常工作，一些用户就可能体验到错误和可访问性问题。

幸好，在绝大多数情况下，内联脚本是完全可以避免的。JavaScript提供了原生的方法来遍历某个文档的标记结构，无须添加属性或编辑标记就能应用事件。

5.1.2　引用外部JavaScript

在HTML里构建和应用格式良好的脚本文件引用很容易，只需在HTML文档的head里为每个文件添加一个脚本元素，并将它的src属性设置成外部JavaScript文件的路径即可：

```
<script src="js/enhancements.js" type="text/javascript"></script>
```

type属性也是说明文件内容类型（在这个案例中是text/javascript）所必需的，让服务器能够正确解释它。

开发社区对脚本文件最佳的输出位置存在着争议：它应该像HTML4规范里那样放在head元素里，与CSS引用并存，还是应该放在文档末尾，让网页内容能够首先加载？

从我们的角度看，这些都不是关键问题。在用渐进增强方法编写代码时，引用外部脚本最重要的问题不是把它们放在*哪里*，而是*何时*引入它们。脚本文件应该在确定浏览器能正确渲染它们后才载入网页。要确定哪个脚本文件能在某种设备上正常工作，必须在浏览器加载它们之前测试这些脚本，然后只输出能够通过测试的脚本增强信息。（下一章会详细描述如何用这种方式加载脚本。）

5.2　理解 JavaScript 在基本体验里的位置

第4章列举了一些我们推荐的“安全”CSS属性，用它们可以对网页的基本体验进行细微的视觉性改善，而且不存在引入可用性问题的风险。

但是，对于JavaScript来说，这样的列表并不存在。当JavaScript遇到某个问题时（无论是因为浏览器支持不足还是开发者使用的逻辑有缺陷），它倾向于“砰”地一声报错，无论错误的严重程度如何都将它呈现给用户。更糟糕的是，JavaScript错误可能会停止进程的运行，让网页停留在部分增强或者完全不可用的状态。

因此，我们建议在基础标记里包含一条JavaScript引用：运行能力测试套件所需最低限度的JavaScript。我们总是用标准HTML静态元素和表单控件创建基本体验（基础标记和安全样式），构建一个无须附带任何JavaScript事件也能在各种设备上运作的可用界面。如果（而且只有）浏览器通过了增强体验要求的全部能力测试，我们才会使用EnhanceJS载入额外的JavaScript，插入增强信息。

5.3 脚本增强的最佳实践

用JavaScript增强网页的方式有很多，从简单的内容显示/隐藏切换，到完全改变用户体验的复杂转换不等。这一节会演示一些最佳实践：使用渐进增强将JavaScript应用到网页，以及给网页现有内容添加增强行为，生成或请求新的标记，应用增强样式和管理内容可视性。

5.3.1 在内容就绪时运行脚本

和CSS不同，JavaScript要求它操作的HTML元素在脚本执行时就已存在。如果某个所需的元素对脚本不可用，浏览器就会提示出错。要避免发生这种情况，JavaScript必须推迟运行，直到标记加载完成。

JavaScript原生的onload事件处理程序，在它的关联元素包含的所有内容（包括图像和iframe等所有附属品）都加载完毕后会触发。在onload事件处理程序里添加一段脚本，可以确保它作用的元素在脚本运行前都已加载完毕并准备就绪：

```
document.body.onload = function(){
  // 对body里的内容进行操作
};
```

在许多情况下，只要有了HTML标记，即使全部网页资源（图像、iframe等）尚未加载完成，脚本也能安全运行。没有任何一种原生的JavaScript方法能在当前所有浏览器里正常工作，不过流行的JavaScript库提供了自定义事件处理程序（例如jQuery的ready方法），基本标记结构就位后可以立即执行某个脚本：

```
$(document).ready(function(){
  // DOM相关的代码放在这里!
});
```

5.3.2 给标记应用行为

在增强体验里，JavaScript控制的主要活动之一是无缝应用行为来对用户的交互作出反应。有两种方式可以将JavaScript事件处理程序连接或绑定到增强标记上。

- **事件绑定**（event binding）能将鼠标点击或者按键等事件处理程序应用到网页里的特定元素上。
- **事件指派**（event delegation）利用了JavaScript的事件反升（event bubbling）功能，将事件监听器应用到父级容器上，而不是将处理程序直接绑定到各个单独的元素上。

1. 使用事件绑定

JavaScript是一种事件驱动的语言，意思是它既可以在载入时执行，也可以“监听”某个事件，在晚些时候才触发它。JavaScript的事件对应环境活动（例如load和error）和用户活动（例如mousedown、mouseup、click、mouseover、mouseout和keypress等）。

JavaScript的事件处理程序是给网页元素应用行为的内建方法。使用事件处理程序将行为绑定到元素上时，该行为的执行会被推迟，直到事件发生。JavaScript的事件监听能力是构建智能和响应灵敏的网络应用程序过程中的一个主要因素。

举个例子，下面的代码展示了如何通过id找到某个锚元素，然后将它和click事件处理程序绑定：

```
var myAnchor = document.getElementById('myAnchor');
myAnchor.onclick = function(){
  //放在这里的JavaScript会在myAnchor被点击时执行
};
```

但是，用这种方式绑定事件监听器会覆盖之前应用到该元素相同事件上的所有监听器。要以不覆盖其他监听器的方式添加事件监听器，W3C的第2级DOM（DOM Level 2）事件模型列出了addEventListener方法。Internet Explorer采用了自己的事件绑定方法attachEvent，这就意味着添加事件监听器的跨浏览器解决方案需要检查并使用受支持的方法。

为了简化对多个事件监听器的绑定，一些JavaScript库提供了自定义方法，例如jQuery的自定义bind方法。这些库还提供了选择器引擎，让开发者可以用CSS样式的语法找到元素，使得通过ID、类名或任何CSS选择器绑定事件变得简单。下面的代码演示了一个接受两个参数的bind方法：一个参数是事件名称（在这个案例里是click，但它也可以接受其他事件名称，比如mouseover和mouseout等，甚至可以接受多个事件），另一个是事件触发时要执行的回调函数：

```
$('#myAnchor').bind('click', function(){
  //放在这里的JavaScript会在myAnchor被点击时执行
});
```

jQuery bind方法回调函数的执行范围，是在与事件绑定的DOM元素之内，这样就可以使用JavaScript关键词this指代该元素。举个例子，点击某个元素时，它的class属性可以使用this关键词进行修改：

```
$('#myAnchor').bind('click', function(){
  this.className = "clicked";
};
```

2. 使用事件指派

将事件直接绑定到元素上很有用，但是当你创建复杂的应用程序，需要让许多元素表现出类似的行为时，更有效率和易于管控的方式是使用事件指派来分配事件。

事件指派的一个优点是，绑定到某个父元素上的事件会自动将行为应用到子元素上，哪怕它们当时还没有出现在网页上。这就使它成为了一种特别有用的技巧，适用于在组件创建后再添加标记的网页里使用脚本增强信息。另外，事件指派应用事件的速度很快，内存使用效率也很高：相对于遍历某棵树上的每个节点来绑定单独的事件，所有程序逻辑可以一次性指派到父级元素上。

事件有自己的原生属性，例如pageX和pageY（表示鼠标坐标）以及target（表示触发事件的那个元素），jQuery赋予了这些属性跨浏览器的通用名称。target属性特别适用于应用事件指派，因为用它可以找到是哪个元素触发了某个事件。

事件指派指的是将某个事件绑定到父级元素（比如组件容器或者body元素）这类操作。嵌套在父元素内的某个子元素触发了某个事件时，该事件会“反升”到父节点并触发一段脚本，此脚本执行时会检查该事件的target是哪个元素，如果此target与我们想要的元素匹配，对应的脚本就开始运行。

传递到某个事件处理器回调函数的第一个参数引用的是它的event对象，内含该事件的所有属性。下面的代码展示了绑定到body元素的一个事件，里面包含了条件逻辑，用于检查点击的对象是否是一个锚元素：

```
$('body').bind('click', function(event){
  if ($(event.target).is('a')){
      // 某个锚元素被点击了！
  }
});
```

jQuery通过它的live方法进一步简化了事件指派，让它变得自动化，原理是将事件应用到文档本身，然后检查事件的target，看它与live方法应用到的元素是否匹配：

```
$('#myAnchor').live('click', function(){
  this.className = "clicked";
});
```

5.3.3 用JavaScript构建增强标记

用JavaScript创建额外的HTML并将其插入到增强体验里通常是必不可少的。举个例子，基本体验里的下拉菜单可能会在增强体验里被转变成一个滑块。因为这个滑块的标记离不开JavaScript行为，而且只会给基础标记带来额外的带宽消耗和代码复杂度，所以它不应该包括在基础标记中。相反，这些新的滑块标记必须在增强过程中插入。

增强专用的标记有两个来源：它既可以由JavaScript根据从基础标记里获得的信息生成，也可以通过Ajax请求从服务器上获取。

1. 使用基础标记作为生成增强标记的向导

只要有可能，用基本的基础标记作为参照生成增强标记是一种良好的实践。

举个例子，想象一下你要从基础标记里的某个原生HTML select元素创建出一个滑块。我们可以用JavaScript来解析每个option元素里的文本，并找到原生select里当前选中的是哪个选项。用这些数据，可以在脚本里生成一个通用的滑块标记“模板”，然后将生成的增强标记插入网页。动态生成内容相比用Ajax请求额外内容要快得多，因为一切都是网页里现成的。另外，通过编写通用的脚本，让它将基础标记作为数据源来生成模板（而不是将内容嵌入脚本），就可以对网站里许多不同的滑块使用同一种脚本逻辑。（第12章和第17章会详细展示这一技巧的两个范例。）

在某些情况下，我们会选择在基础标记里编码数据，这样增强脚本就能利用它。举个例子，用于生成工具提示的外部内容的位置可以编码到data属性，增强脚本会用它生成一个Ajax请求。HTML5里的data前缀可以添加到任何字串前面，从而创建出符合Web标准的自定义属性。

极少数情况下，在脚本里存放一小段内容片段或者标记的确是最佳做法。举个例子，请考虑某个对话框标题栏上“关闭”按钮所用的文字。虽然该对话框的主要内容可以来自于基础标记或者从服务器请求到的一份单独文件，但它的标题栏控件和“关闭”按钮存在的原因只是为了使用对话框本身。基础标记里没有这些内容的位置，而且从性能的角度看，向服务器发送一个请求仅仅为了获取这两个字，这是非常低效的。在这样的特定情形中，我们相信把内容放进JavaScript说得通。这样做时，我们会小心地将这段文字存放在一个可配置的JavaScript变量里，并在方便的位置定义这些变量，比如JavaScript文件的顶部。

2. 用Ajax请求额外内容

某些情况下，用Ajax给网页添加额外的标记比在基础标记里保存要更为合理，原因有网页尺寸限制或速度优化目的，避免额外的标记弄乱基本体验，或者其他商业理由。

Ajax这种技术一开始是Internet Explorer 5的一项私有功能，之后被采纳成为Web标准，集成进了大多数现代浏览器中。它让JavaScript从服务器请求数据来更新部分网页，而不必非得重新载入整张网页。

用Ajax从服务器请求内容是一个直观的过程：某个脚本发送一个对HTML特定片段的请求到服务器；服务器返回一段响应文本，可以是纯文本、XML、JavaScript（JSON）或者HTML格式；脚本将文本插入到网页中。

jQuery提供了几种实现Ajax功能的方法，如下所示。

- `ajax`方法。它是最为健壮的，提供了一个对应浏览器`XMLHttpRequest` API的标准化接口，还有多种jQuery独有的事件和属性。
- `get`和`post`方法。它们提供了实现`ajax`方法的快捷方式，用更简洁的配置生成请求。
- `load`方法。它类似于`get`，不过你会在某个DOM元素上调用它，意图是让返回的内容插入到该元素中。

`load`方法还有一个额外的好处：指定过滤条件，请求响应内容的一个子集。举个例子，下面的代码通过指定一个CSS选择器（`#latest`）只请求了某张网页（news.php）的其中一部分：

```
$('div#news-ticker').load('news.php #latest');
```

用这种方式载入HTML片段特别有助于获取一张大型网页里的一小段内容子集。

用Ajax请求内容的一个潜在威胁是服务器响应所需的时间，如果网络较慢或者请求太多，它就可能导致用户界面出现明显的延迟。从服务器取回数据时，良好的做法是在请求执行过程中显示一个“加载中”的指示器。

3. 确定内容何时应加入基础标记

大多数情况下，所有的内容和功能都应该在基础标记里展现出来。但是可能有时某个特定功能过于复杂，在缺乏JavaScript行为时难以使用。举个例子，第2章提到，给照片管理器在基本体验里重建增强版的照片裁切工具（使用时用户需要指定裁切点的精确像素坐标）使用难度太高。考虑到其复杂程度，像这样的案例最好还是留给增强体验。

另外一些情况下，JavaScript禁用时内容可能会毫无意义。举个例子，“打印本页”这个链接

完全依赖JavaScript才能工作，所以当JavaScript不存在或被禁用时它给用户的感觉就是损坏的。类似地，某个需要JavaScript才能工作的自定义表单控件标记，在无JavaScript的环境里只会让用户感到困惑。这些功能类型的标记应该排除在基础标记之外，只在增强体验里用JavaScript添加进来。

5.3.4 管理内容可见性

用JavaScript给增强页面添加额外的内容和标记时，你需要做个判断，原来的基础标记是否应该保持可见。

有时增强内容的角色是基础标记的补充，让两者能在网页上有意义地共存。请考虑一个基于canvas的图表，它根据基础标记里某个HTML数据表格生成：视觉性的图表外观和细节性的数据表格可以共同呈现，为用户提供额外的丰富性和意义，所以我们肯定会考虑在网页上同时显示两者。类似地，如果将某个input或者自定义的select元素增强成一个滑块，通常会将原生元素和增强滑块同时放在页面上，给用户提供更多样的交互机会。

然而，有些时候增强内容完全取代了基础标记里的版本，比如用自定义样式的select元素取代了原生版本的控件。让它们同时可见不但多余，还可能会让用户感到困惑。在这些情况下，将原生控件对用户隐藏起来十分重要。

1. 从视觉上而非听觉上隐藏内容

考虑显示和隐藏时，我们总是会考虑广大用户，包括那些使用屏幕阅读器的用户。前面提到，当我们根据HTML table数据用HTML5的canvas元素创建图表时（将在第12章详细介绍），图表和表格是两种有点重复的内容。既然它们是重复的，有些设计师可能会选择隐藏数据表格，显示图表。但是图表只对视力正常的用户有用，对使用屏幕阅读器的用户来说，从网页中彻底移除包含数据值的表格会导致内容不可访问。

如果隐藏对屏幕阅读器用户来说仍然是必需的内容时，最重要的是避免使用display: none或visibility: hidden等CSS属性。它们不仅会将内容从视线里隐藏起来，还可能会波及到屏幕阅读器。为了安全地隐藏内容，应该对它使用绝对定位并设置一个很大的负数左边值（position: absolute; left: -99999px;），使它可靠地移出屏幕。

2. 对所有用户隐藏内容

一些自定义表单控件在增强体验里生成的内容与基础标记执行相同的本质功能。在网页上同时保留原生和增强版的控件，对所有用户来说都是个麻烦。拿某个自定义样式的下拉菜单为例（会在第17章详细介绍），它使用a、ul、li和div等元素创建出一种对CSS友好的可用版本来代替原生的select元素。如果增强版控件生成并插入页面，基础版的select元素就完全多余了，同时看见这两个可能令人非常困惑。

许多设计师/开发者会用JavaScript把原生控件从网页中彻底移除，只剩下增强版的控件。可惜，这可能会带来很大问题：新版的标记虽然在选择功能方面完美可用，但是没有办法将它的数据和表单其余部分一起提交，因为归根到底它只不过是一堆div和一张列表。

在这个案例中，有必要创建一个代理，也就是用`display: none`从视觉上隐藏屏幕里的原生表单控件，但仍然将它留在网页标记里，保存值用于表单提交。然后，用JavaScript连接增强版替代控件和它对应的原生元素，这样当用户修改控件值时程序会自动操作隐藏的`select`元素。如果代码编写得当，代理就能让表单用原生控件的值进行提交。对服务器来说，就像增强版控件不存在一样。

5.3.5　应用样式增强

通过加载和操作CSS给HTML标记添加样式是创建增强体验的基础。JavaScript帮助实现增强的方式有许多 。

- **动态加载额外的CSS文件**。JavaScript可以用来将样式表载入页面。这是我们所用方法里的一个关键要素：我们已知一些复杂样式在某些浏览器里的支持程度不一，因此将它们存放在外部样式表里，首先用EnhanceJS测试某种浏览器是否能成功渲染出增强样式，符合特定条件后再插入一个或多个增强样式表。这一技巧对那些需要应用复杂样式的大规模增强是很有用的，比如将一张线性网页转换成多列网格布局。
- **切换HTML类**。我们提倡使用JavaScript来添加、移除和切换CSS类，而不是用JavaScript编写内联的CSS规则。因为样式规则存放在外部CSS文件里，而非遍布整套脚本逻辑，所以我们能够清楚地分离样式和行为，简化调试和维护工作。JavaScript还提供了一种方法，为那些不能原生支持伪类（例如`:hover`或`:focus`）的浏览器提供了CSS支持扩展。举个例子，利用JavaScript，能在用户把鼠标放在某个元素上方时动态指派一个“悬浮”类。
- **用`style`属性内联应用样式**。在极其有限的情形中，操作CSS类是不现实的。举个例子，当JavaScript需要动态设置或者为样式规则添加动画效果时（此时可能需要对对话框窗口的位置和尺寸属性进行数百次操作，来实现它在屏幕上的动画效果、重设大小和重定位），创建一组数量固定的静态类是不现实的。在这种情况下，有限度地使用动态内联样式设置则是最有效的选择。如果某个动画需要进行动态样式脚本编程，比如实现同时淡入淡出（cross-fade）效果，另一种好的做法是当动画结束时移除所有的内联样式，然后用类属性替换它们，以保持视觉外观并避免将内联样式遗留在网页中。最后，如果某些案例里的标记被用来视觉化呈现数据（例如给某个`div`元素添加样式，让它表现为条形图里的长条），一种现实的做法是使用JavaScript生成一个内联样式并将其应用到该`div`上，这样它的外观就与数据大小相匹配了。

5.4　保持和增强可用性与可访问性

用语义化HTML构建的基本网页一般来说是天然可访问的。屏幕阅读器和其他辅助技术被设计用来理解HTML，并将它的的内容传达给用户。基础标记很少需要在这些环境里进行测试。大多数情况下它都好用。

可惜，对增强体验而言，这一规律并不是经常适用。即使增强信息是由最具描述性的标记构

建的，也可能会遇上HTML就是没有足够的元素来描述各项功能的情况。一个视力正常的用户能轻易辨认出某个自定义样式的下拉菜单是原生select的等价物。与之相反，屏幕阅读器只能通过构建这一控件的元素（一个标准HTML链接、一两个div和一张无序列表）来识别它，而这几乎不能说明它在用户界面里的角色。幸好，ARIA属性能帮助缓解这一问题（至少是对那些使用现代屏幕阅读软件的用户而言），本节稍后进行讨论。

用户辨认出一个熟悉的控件时，他们通常会认为自己能使用熟悉的传统键盘操作，比如用空格键打开一张下拉菜单，用方向键在选项里导航，以及再次按下空格键做出选择。创建自定义控件时，开发者通常有责任手动移植这些HTML控件原生的行为。构建可用键盘访问并对屏幕阅读器有意义的自定义控件，需要付出额外的努力和编程时间，可惜这类控件的构建工作却常常留到最后一刻草率实现，或者干脆彻底排除在最终产品之外。

这是一个大问题，不仅对残障用户来说如此，对那些自主选择（或者被限制只能）通过键盘导航浏览网站的人来说，也是如此。缺乏键盘快捷命令功能的自定义控件相比它们的原生等价元素有所退步，还会让用户感到不爽，乃至于抵消了增强所带来的好处。

从这些原因看，重要之处在于自定义控件要能代替原生控件的功能，不仅要支持空格键、回车键和方向键，还要支持向上翻页键（Page Up）、向下翻页键（Page Down）、起始键（Home）和结束键（End），等等。这些功能需要额外的时间和高度的注意力才能开发得又好又准确（这就是我们建议只要有疑虑就应优先考虑使用原生控件的原因）。但是，但在某些情况下构建自定义增强控件物有所值，因此十分有必要将原生控件所有的可访问性和可用性功能囊括到自定义组件里来。

5.4.1 实现键盘访问

开发自定义组件时，要点在于首先要复制原生的功能，然后再用额外的功能扩展它。这一点对键盘事件来说特别重要，因为用户对与某个熟悉控件的交互方式抱有预先形成的期望。这一节会讨论键盘访问的两个主要考虑对象：制表键（Tab）焦点和编写事件脚本。

1. 管理制表键焦点

在所有HTML元素中，只有少数几个（包括a、input、button、select和textarea）可以获得原生键盘焦点。焦点由制表键进行控制，反复按下这个键会让焦点在网页上的有效元素之间移动。

在基本网页里，元素焦点移动的顺序被称为*制表键顺序*（tab order）。这个顺序可以用tabindex属性控制，该属性接受两个值：0让某个元素能够获得焦点，根据它在默认源代码里的顺序指定它的制表键顺序；-1会把某个元素从制表键顺序中彻底移除。（虽然从技术上说可以指定一个正整数，覆盖默认的制表键顺序，但是不推荐这么做，因为如果做得不正确可能会引发可用性问题。）

在增强版的网页里管理焦点需要用到JavaScript，使自定义控件能够模拟其对应原生元素的行为：具体而言，要点在于要编写出一套逻辑，让控件可以通过制表键获得焦点，并在获得焦点后根据设计意图立即启用空格键、回车键、方向键等按键在控件中导航。类似地，按下制表键应该可以将焦点从一个控件转到下一个控件。

自定义控件里的某些元素有时会具备我们不想复制的原生焦点能力。举个例子，在自定义样式下拉菜单里，我们用一张无序列表加上一组锚链接（它们原生能够获得制表符焦点）组成可选项。这种情况下，我们希望这些锚点的行为类似于标准下拉列表的option值，后者原生可以使用方向键访问。我们可以将每个链接（a）的tabindex设置为-1，防止它获得制表键焦点。要点是，如果某个元素的tabindex设为-1，那么它的焦点就只能通过程序进行管理了，要用JavaScript。（第17章会详细演示如何管理自定义下拉菜单控件的焦点。）

2. 编写键盘事件脚本

JavaScript里的许多键盘事件（例如keydown、keyup和keypress）可以绑定到元素上。绑定了某个键盘事件，就可以使用条件逻辑来判断按下的是哪个键，以及触发哪些操作。

键盘事件提供了一个名为keyCode的属性，它储存一个数字，对应按下后触发事件的那个键。（Keycode数字的参考资料在网上免费可查。）下面的代码演示了用jQuery将keypress事件绑定到某个锚点上：

```
$('#myAnchor').keypress(function(event){
  if(event.keyCode == 39){
    //按下了右方向键
  }
}
```

基于这种逻辑，可以将自定义组件的功能映射到对应的键盘操作上。举个例子，之前的代码范例将keypress事件绑定到了一个锚点上，按下右方向键时会执行一段脚本。这段特定的代码在某些时候很有用，比如给某个树形控件添加键盘支持，将焦点移至树节点，然后用方向键切换控制其子元素的可见性。

5.4.2　指派WAI-ARIA属性

键盘支持被集成进自定义控件后，这个控件对所有用户的可访问性就大大提升了。但是，仍必须正确通知屏幕阅读器某段标记正在扮演一个自定义组件的角色，而不是仅仅在那里组织内容。为此，我们使用W3C WAI-ARIA规范（通常简称ARIA）里概述的一些属性。

第3章介绍了ARIA，并且讨论了在基础标记里给元素应用角色，如何能为屏幕阅读器用户开启一套逻辑导航选择。ARIA属性还能扮演一个重要的角色，帮助辨识和描述增强体验里的动态内容。举个例子，可以用标准HTML锚链接制作一个可点击的“打印”按钮，通过给它应用一个值为button的ARIA role属性，就能恰当地将它描述为一个按钮。

大部分ARIA属性及其关联的值本意是为了在增强体验里使用，一般也都会通过JavaScript进行应用。和所有HTML属性一样，ARIA属性可以通过JavaScript原生的setAttribute方法进行设置或修改。下面的代码演示了给某个锚链接指派一个button角色：

```
document.getElementById('myAnchor').setAttribute('role','button');
```

使用jQuery则代码长度大大缩短：

```
$('#myAnchor').attr('role','button');
```

5.4.3 测试可访问性

增强体验经常会背离并改变原生行为，所以关键是要用屏幕阅读器等辅助技术对它们进行测试。开发者不仅需要确保他们的代码在技术上与这些设备兼容，还要让网页在用户的视角里具备意义。

一些最流行的屏幕阅读器十分昂贵，以至于让许多开发者负担不起。幸好，有许多类似的阅读器是免费的，并且功能毫不逊色，就可用性测试而言十分有用。

- Mac操作系统自带一个非常健壮的内建屏幕阅读器，名为VoiceOver。你可以打开“系统首选项”（System Preferences），找到“通用访问”（Universal Access）版块，然后在那里启动它。（“通用访问”版块里还有其他许多有用功能可以用来测试可访问性，比如可以将屏幕切换到灰度模式，检查对比度是否合适。）
- 对Windows用户和在PC上进行测试的开发者，我们推荐下载和安装免费的NVDA屏幕阅读器，地址是http://nvda-project.org。

提示 这里有一篇介绍安装和使用NVDA的优秀文章：http://t.cn/zRlTbxA。

- JAWS是网上最流行的屏幕阅读器之一，它有一个免费的40分钟演示版。这个版本用起来稍微有些不便（使用40分钟后，它会要求重启Windows），但鉴于这个广泛使用的屏幕阅读器的流行程度，我们仍然认为这是值得的。（在Filament集团里，我们给Mac安装了VMWare Fusion，让Windows和其他应用程序共同运行，所以我们能测试上面提到的全部三种屏幕阅读器。）

除了用屏幕阅读器进行内部测试，另一种了解网站是否可访问的宝贵方法是，请残障用户现场进行可用性测试。*Just Ask: Integrating Accessibility Throughout Design*的作者们维护了一个非常有用的资源网站，里面有各种帮助性提示、指南和其他资源：www.uiaccess.com/accessucd/ut_plan.html。

5.4.4 维持状态和“后退”按钮

Ajax应用程序（而且特别是Ajax应用程序）引入了一个独特的可访问性问题：随着用户与网页进行交互，屏幕上描绘的内容和浏览器地址栏里显示的网页状态之间可能会失去联系。Ajax调用并不原生记录在浏览器历史里，所以，产生的变化可能与用户对他所执行操作的认知并不相符。

决定是否要在浏览器的历史记录里追踪网页变化时，一个不错的经验法则是，看用户是否可能认为他们的操作将他们带到了一张新的“页面”。举个例子，如果一个基于JavaScript的标签页横条控制了网页里足够大的区域，感觉它就像是一个主要导航元素，那么可能就值得通过历史追踪每一次的标签点击，这样用户就可以通过标签视图的历史记录或者收藏那一页，回退到之前的页面。追踪还可能适用于搜索结果页，用来在结果分组之间导航，或是在一组标签中确定自己的位置。

在Ajax应用程序里追踪状态需要用JavaScript更新和关注URL#号串（它是网页URL的一部分，能通过ID属性链接到文档的某个版块）发生的变化。举个例子，下面的URL会将页面滚动到ID为content的元素上：

http://example.com/#content

9.3节会分析如何制作一个插件，让它用URL#号串在Ajax应用程序里管理历史记录。

我们已经分析了如何无缝应用JavaScript，如何编写语义化HTML，以及清楚分离和编写有效的样式，现在你的手头上已经有了渐进增强的所有核心要素。接下来，我们会向你展示如何用能力测试将它们组合在一起，确保你只对有能力的浏览器输出增强信息。

第 6 章

测试浏览器能力

6

如果某种设计包含高级的CSS和JavaScript，为了确保你能提供可用的体验，借助渐进增强方法进行开发（用语义化HTML构建内容和功能，然后叠加上更为复杂的样式和行为）就是有力的第一步。但是，单纯把标记从表现和行为中分离出来，并不能保证增强信息只输出给有能力渲染它们的浏览器。

幸好，有一种方法可以让你免于猜测：测试浏览器能力。通过测试某个浏览器实际上能做什么，你就能更明确地判断它是否会正确渲染页面增强信息。

在Filament集团里，我们将此看做用渐进增强方法进行开发最重要的一个方面。我们的能力测试会同时检查JavaScript和CSS的支持情况，确保一系列关键功能可以正确渲染，这样就能够假定设计所需的那些特定样式或行为的依赖关系会按照设计的方式工作。我们的经验是，大多数复杂界面和交互组件都依赖于JavaScript和CSS协同工作。因此，我们构建了EnhanceJS。它是一套测试框架，能够检查能力并有条件地应用增强，另外还添加了一个备用选项，以防能力测试出现不准确的情况。

本章会分析EnhanceJS的构建方式和工作原理，解释如何在你自己的项目里使用EnhanceJS，并介绍如何扩展EnhanceJS里许多可定制的手段，从而更好地控制增强信息的输出方式。本章最后会讨论如何进一步利用能力测试的这些原则创建你自己的测试。

6.1 EnhanceJS：一套能力测试框架

EnhanceJS是一套轻量级JavaScript框架，在某人首次访问网站时运行一套脚本和样式能力测试。如果所有测试都通过了（就是说，如果测试确认浏览器确实支持所有功能），它就把CSS或者脚本增强添加到文档上。如果任何一项测试没有通过，那就不会有增强，网页会按照原样显示：一种功能齐全的基本体验。

测试通过后，这个脚本通过两种方式给网页应用增强。

(1) 它给`html`元素指派一个名为`enhanced`的类。附带样式表里所有属于该类的样式规则在指派类时生效。

(2) 如果指定了任意数量的脚本或样式表文件，它们会被添加到文档的`head`里。

随后，EnhanceJS将浏览器是否通过测试的结果保存在一个cookie里，避免此框架在每次载入

网页时都运行一遍能力测试。EnhanceJS在后续页面中运行时，它首先检查那个cookie，接着会添加增强信息（或者不添加）。如果没有找到cookie，它就会再次运行那一套测试。

作为一种备用选择，这个框架脚本会将一个切换链接添加到文档主体中，让用户可以手动切换网站的基本版和增强版。这个级别的用户控制在一些场合很有价值，比如某种浏览器通过了测试，应用了增强，但是有东西不能正常工作。虽然我们测试浏览器能力来防止出现这种情形，但是用户在增强体验里遇到问题的可能性总是存在的（特别是考虑到如今的可上网设备多种多样）。能够将网页切换回更简单的、功能完备的基本体验，是我们渐进增强方法论里的一个关键功能。如果有用户喜欢浏览更简单而且速度可能更快的网站版本的话，这个功能也很顺手。

EnhanceJS框架由3个主要部分组成。

- 写入每张网页的一小段脚本块会调用`enhance`函数。此函数接受若干选项，包括如果浏览器有支持能力的话应该添加哪些JavaScript或CSS文件。如果JavaScript没有启用，这个函数就会被忽略，使浏览器渲染出基本版的网页。
- 存放在一个单独文件（enhance.js）里的框架脚本。它是整套操作背后的大脑：它运行全套能力测试，记录哪些通过哪些未通过。如果所有测试都通过了，它就把`enhanced`类指派给`html`元素，如果指定了文件的话还会把它们插入到网页里。
- 一个接入框架脚本的套件，内含一个或多个能力测试。套件里的每个测试都针对单个JavaScript方法或者CSS属性，如果浏览器支持此功能，该测试就返回一个`true`值。默认情况下，我们加入了一些涵盖JavaScript和CSS代表性功能的测试，可以进行编辑或扩充，以满足具体项目的需要。

能力测试提供了一套机制，使增强信息只面向能够处理它们的浏览器。

能力测试框架

测试浏览器能力是一种新兴技术，但值得高兴的是，我们已经知道至少有另外一个库能够补充和扩展EnhanceJS，这个库是Modernizr（http://modernizr.com），它为HTML5里的许多功能提供了详细的测试案例。随着对渐进增强和能力测试的讨论进一步深入，我们计划留意那些新的库和技巧。请查看本书的网站（www.filamentgroup.com/dwpe）了解另外一些有用的能力测试框架及其链接，我们会试着用更有用的资源更新这个网站，只要我们能找到。

EnhanceJS的机制：测试是怎样工作的

EnhanceJS框架使用两种测试机制来判断浏览器支持：对象探测用于JavaScript功能，一个基于脚本的自定义方法用于测试CSS渲染的准确性。

对JavaScript功能支持而言，久经考验的对象探测方法提供了一种方式，来判断某种浏览器是否支持特定的JavaScript对象、方法或属性，只需简单地将它放入条件语句内，然后运行即可。

举个例子，`getElementById`是一种常用的方法，它使用某个元素的`id`值来在DOM里找到它：

```
var myEl = document.getElementById('myElement');
```

我们可以用以下方式测试浏览器，看它是否支持getElementById这样的方法：

```
if (document.getElementById) {
  // 浏览器支持此对象
}
```

我们在EnhanceJS里用这个简单的对象探测模型来测试JavaScript的支持情况，网页调用enhance()时会测试下面这些能力。

- document.getElementById：通过ID找到元素。
- document.getElementsByTagName：通过标签名找到元素。
- document.createElement：创建新的元素。
- xmlHttpRequest：测试Ajax支持（我们会测试此对象的几种流行实现方式）。
- window.resize：探测窗口大小调整时产生的变动。
- window.print：触发浏览器的打印对话框。

在默认的EnhanceJS测试套件中包含针对这些对象的测试，是因为我们发现它们能够很好地代表一类最常见操作，支持这些操作，才能成功访问和操作文档元素。（本章稍后会讨论如何修改这张列表，配置EnhanceJS，使它适用于你自己的项目。）

测试CSS的支持情况则稍微有些复杂，原因有两个。

- CSS出问题时，它会静静地出。如果一种浏览器能理解某个CSS属性，它就会尝试渲染这个样式，哪怕只能部分渲染；如果不理解，它就会忽略这个属性及其内部属性，而不会引发错误。虽然这种优雅的失败机制是使用CSS的好处之一（它让我们可以安全地试验新CSS属性，哪怕它有时不能正常工作），但是它也会让CSS支持变得特别难以测试。
- 即使有可能像测试JavaScript对象那样测试CSS属性的支持情况，但这样做就不能准确了解它实际上是如何渲染的，因为浏览器对CSS的支持程度存在巨大的差异（从视觉上准确和可预测，到稍有不足，再到存在严重问题。）举个例子，某种浏览器也许会报告它支持margin，但渲染实际元素的外边距时，可能会比样式表里指定的值更大或者更小。

虽然没有一种方法能够简单可靠地询问浏览器是否支持某个CSS属性，然后获得一个能确保良好体验的答案，但是我们发现的确有种方法可以测试CSS是如何在网页里渲染的。我们开发了许多测试脚本，它们会创建一个HTML元素，给它加上样式并插入文档主体，然后手动检查各种属性，从而判断浏览器是否正确地渲染了它们。

举个例子，这些测试其中的一种会检查浏览器对CSS盒模型（box model）的渲染情况。盒模型规定一个盒子（或者块级元素）的实际宽度是它的CSS宽度、边框和内边距属性之和。在那些不能正确支持的案例里，它可能会产生级联效果（cascading effect）而实际上破坏页面布局。EnhanceJS会创建一个div，应用1像素的宽度和1像素的左右内边距（内边距总共2像素）。然后，它将实际宽度值与计算总和加以比较，如果盒模型的实现方式正确，结果应该等于3像素：

```
// 检查盒模式支持情况的函数
function boxModelSupported(){
  var newDiv = document.createElement('div');
  newDiv.style.width = '1px';
```

```
    newDiv.style.padding = '1px';
    document.body.appendChild(newDiv);
    var divWidth = newDiv.offsetWidth;
    document.body.removeChild(newDiv);
    // 比较测量值和预期值 (3)
    // ===既比较类型又比较值
    return divWidth === 3;
  }

  // 检查函数返回的结果
  if (boxModelSupported()){
    //支持盒模式
  }
  else {
    //不支持盒模式
  }
```

除了盒模型测试之外，EnhanceJS还采用了类似的函数来测试几种已知在广大新旧浏览器里支持程度不一的CSS属性。

- position：元素在页面里的位置。
- float：浮动元素，使其相邻排列。
- clear：清除某元素周围浮动的同级元素。
- overflow：控制溢出内容的渲染方式。

每一种测试能力的方法（JavaScript对象探测和测试CSS渲染准确度）各自都能良好地工作。但是，经验告诉我们，许多复杂的界面会将高级JavaScript和CSS功能组合到一起，而不是单独应用它们。因此，我们打包了这些测试，在网页载入时同时应用它们，然后使用总体结果来判断是否能安全添加增强信息。

6.2 通过 EnhanceJS 应用增强

要运行EnhanceJS，以下脚本引用必须存在于网页之中，放在head元素里面则更好，确保测试能立即开始（这样就降低了网页在增强信息添加之前就显示出来的可能性）。

- 对enhance.js脚本文件的引用。请在http://enhancejs.googlecode.com上下载最新版的enhance.js，将其复制到你项目的脚本文件夹里，然后将脚本引用添加到网页头部，就像这样：

```
<script type="text/javascript" src="js/enhance.js"></script>
```

- 在head里位于文件引用之后的一个脚本代码块，调用了enhance功能：

```
<script type="text/javascript">
  enhance();
 </script>
```

当某种浏览器通过测试套件后，EnhanceJS将enhanced类添加到html元素上，我们可以用它来界定样式属性，从而叠加上增强样式。

举个例子，当EnhanceJS测试通过并确认存在CSS float属性支持时，在基本体验里组织成一列的内容就可以转换成增强体验里的多列布局。

为此，应该给所有增强版CSS选择器加上html.enhanced前缀，后跟一个空格。带有这个前缀的规则只会应用在属于enhanced类的html元素的所有子元素上。

不带有这个前缀的规则会同时应用到基本体验和增强体验里（如图6-1所示）：

```
div.contentA,
div.contentB {
  border 1px solid #000;
}
```

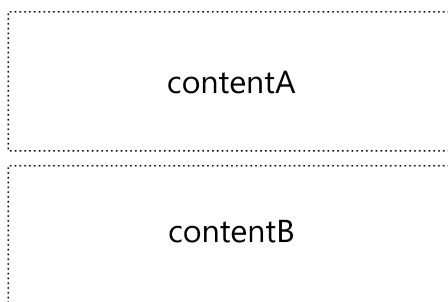


图6-1　应用基本样式后，这些div是以竖直方式堆叠的

而带有此前缀的规则只有EnhanceJS测试通过后才会应用到网页上（如图6-2所示）：

```
html.enhanced div.contentA,
html.enhanced div.contentB {
  width: 50%;
  float: left;
}
```

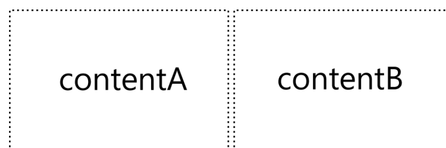


图6-2　应用enhanced类后，这些div就并列浮动了

对那些相对简单、增强样式不多的网站而言，我们可能会把这些样式放在基本样式表里，从而无须进行额外的服务器请求就能立即使用它们。但是，为每个增强元素都单独添加这些选择器，会给基本体增加额外的代码，到一定程度后这种即时优势就被增加的文件尺寸给抵消了。

对那些更为复杂的增强或大规模页面转换而言，我们建议配置EnhanceJS，将外部的CSS或JavaScript文件附加到页面。用这种方式附加文件能确保那些使用低能力设备的用户，能够获得加载速度更快的基本体验，此外还可能为网站主机节省一些带宽，因为增强文件只会发送给能够渲染它们的浏览器。

防止无样式内容闪现

通过渐进增强方法制作的网页可能会遇到一种称为“无样式内容闪现”（flash of unstyled content，简称FOUC）的现象，表现为网页在没有应用CSS时载入，随后快速转变成带有样式的原定布局。当CSS在网页开始渲染后才动态加载和应用，或者当JavaScript操作页面某个正在加载的部分时（导致内容重绘或重排），这种现象就会发生。

幸好，EnhanceJS提供了一种方法来对用户隐藏无样式内容，直到CSS加载完成。EnhanceJS在测试通过时给html元素添加了一个enhanced类，我们可以利用这个类，在增强体验里的内容加载时隐藏它，方法如下。

(1) 在随网页输出的基本样式表里，用enhanced类的一条内部规则隐藏所有的body内容：

```
html.enhanced body { visibility: hidden; }
```

(2) 在网页最终样式表的尾部添加一条规则：

```
html.enhanced body { visibility: visible; }
```

因为enhanced类在测试通过后会立即应用到html上，所以网页会保持空白，直到所有样式都加载完成。

另一种速度更快的FOUC应对方法是把EnhanceJS集成到服务器端的脚本里，对其进行优化，本章后面会讨论这种方法。

6.3　配置 EnhanceJS

创建EnhanceJS时，一个重要的目标是让它具备灵活性，这样开发者就可以配置它，使它按照他们的网站或者应用程序需要的方式工作。我们让enhance函数接受一组预先定义的选项，以对象格式作为参数传递，这样就提供了一套干净的机制，可以指定许多键/值对（请注意括号里的大括号，以及每个对后面的逗号，除了最后一对）：

```
enhance({
  key1: value1,
  key2: value2,
  key3: value3
});
```

EnhanceJS为以下行为提供了可配置的选项。

- 载入额外的样式和脚本文件。
- 改变体验切换链接的外观，或者禁用它。
- 当浏览器未通过某项测试时抛出一个alert（用于测试目的）。
- 指定你自己的额外测试。

下面几节会详细描述如何以及何时指定这些选项。

6.3.1　载入额外的样式表

我们经常会设计和开发应用程序样式的界面，它们需要数量众多的增强样式，将结构轻巧的基本HTML转换成更加复杂的网格布局和交互组件。我们建议将用于扩展基本体验样式规则的增强样式存放在外部样式表里，并用EnhanceJS来添加它们。

要载入一张额外的样式表，需要在loadStyles选项后面指定一个带有引号的文件路径：

```
enhance({
 loadStyles: [
     '/css/enhancements.css'
 ]
});
```

loadStyles选项接受一个包含若干值的数组（记录在围绕文件路径的方括号里），这就意味着你可以指定多个样式表。每个样式表必须单独列在一组引号里，多个样式表之间用逗号分隔：

```
enhance({
  loadStyles: [
      'css/enhancements.css',
      'css/enhancements2.css',
      'css/enhancements3.css'
  ]
});
```

提示　不要在数组最后一个样式表的后面加逗号，这么做会产生一个JavaScript错误。

EnhanceJS测试通过后，它会为每一个指定的样式表各创建一个链接元素，每个链接元素带有指定的文件路径，然后将它们根据列出顺序添加到文档头部：

```
<head>
<meta http-equiv="content-type" content="text/html;charset=UTF-8">
  <title>自定义输入</title>
  <link href="css/basic.css" type="text/css" rel="stylesheet">
  <script type="text/javascript" src="js/enhance.js"> </script>
  <script type="text/javascript">
  // 运行能力测试
  enhance({
      loadStyles: [
          'css/enhancements.css',
          'css/enhancements2.css',
          'css/enhancements3.css'
      ]
 });
 </script>
 <link href="css/enhancements.css" rel="stylesheet" type="text/css">
 <link href="css/enhancements2.css" rel="stylesheet" type="text/css">
 <link href="css/enhancements3.css" rel="stylesheet" type="text/css">
</head>
```

1. 指定样式表属性

可以用许多属性对样式表引用进行限定，这些属性会将额外的意义赋予样式表。下面列出了常用的属性。

- title提供了一段简短的样式表介绍，在指定备用或首选地位时则是必需属性。
- media让样式面向某种特定的媒介格式，比如打印、屏幕（指计算机屏幕）或手持设备。
- rel标识出为网页指定的文件关系。它的默认值是stylesheet，不过还可以指定alternate stylesheet来赋予某个样式表备用地位。

注意　请回顾第4章里的CSS部分，那部分更加详细地讨论了分配样式表属性的最佳实践。

为了适应具体的属性，loadStyles选项还接受JavaScript对象表示法形式的样式表引用，这就意味着文件路径和所有额外属性要用逗号分隔的键/值对在大括号里指定。

举个例子，如果要给增强版网页附加一个打印样式表，那么引用这个打印样式表的方法是提供href属性与文件路径，接着是一个media属性和一个print值：

```
enhance({
  loadStyles: [
      'css/enhancements.css',
      { href:'css/enhanced-print.css', media: 'print' }
  ]
});
```

2. 为Internet Explorer应用条件样式表

Internet Explorer有一些独一无二的CSS渲染bug。为了解决它们，微软公司引入了条件注释标签，利用这个标签，开发者可以创建样式，使这些样式只针对特定（或者全部）版本的Internet Explorer。

我们在EnhanceJS里保留了这个让样式表面向Internet Explorer的能力，方法是提供一个名为iecondition的属性，它可以传递给enhance函数，并附带一个具体的版本号（或者用all这个值来将所附样式表指向全部版本的IE浏览器）。它在脚本引用里的表示方法非常类似于指定样式表属性的方法，但各个版本号并不是在引号里列出：

```
enhance({
  loadStyles: [
      'css/enhancements.css',
      { href:'css/ie-fixes.css', iecondition: 'all' },
      { href:'css/ie-6-fixes.css', iecondition: 6 }
  ]
});
```

6.3.2　载入额外的脚本

EnhanceJS测试通过时，可以指定JavaScript文件并将它们附加到页面中，方式基本上和附加CSS文件一模一样。只需使用loadScripts选项并编写一个数组，将一个或多个由逗号分隔的带引号文件路径放入数组即可：

```
enhance({
  loadScripts: [
      'js/jquery.js',
      'js/enhancements.js'
  ]
});
```

EnhanceJS会根据脚本文件的指定顺序加载它们，因此要点是先列出所有的被依赖脚本。举个例子，当你引用一个脚本库（比如jQuery或YUI）时，要确保首先列出它，这样EnhanceJS就能够先加载它，然后再加载那些依靠库才能正常工作的自定义脚本。

如果脚本文件之间没有依赖关系，就可以加速这个加载过程，方法是将一个名为queueLoading的选项设为false。这么做会禁用默认的JavaScript文件队列，或者说顺序加载，让多个脚本可以同时加载（在浏览器允许的范围内）。

6.3.3 自定义体验切换链接

默认情况下，EnhanceJS会向网页底部添加一个链接，让用户可以手动在基本体验和增强体验之间切换。它给这个链接指派了一个类（默认名称是enhanced_toggleResult），使得用CSS修改它的样式，或者用额外的JavaScript操作它成为可能。因为这个类名在基本体验和增强体验里是一致的，所以可以将下面的样式规则添加到你的样式表里，统一修改它的外观样式：

```
.enhanced_toggleResult { /* 这里放修改切换链接样式的CSS代码 */ }
```

EnhanceJS将切换链接插入body元素的尾部，但是你可以将这个链接移动到网页的其他位置，方法是编写并加入一小段脚本。举个例子，下面的脚本使用jQuery语法找到切换链接，并重新将它添加到网页的另一个元素之中（在这个案例里是一个id为myFooter的div元素），如图6-3所示。

```
$('.enhanced_toggleResult').appendTo('div#myFooter');
```

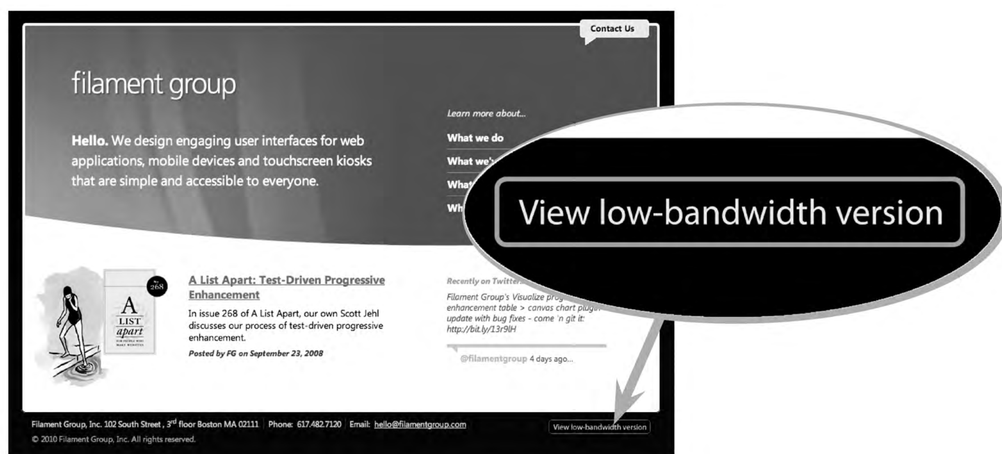


图6-3 页面底部一个修改了样式的切换按钮，供用户“自愿退出”增强体验

切换链接的文字（就是用户在网页上看见并点击的文字）也可以调整。默认情况下，基本版和增强版网页里的链接分别为“查看高带宽版本”（View high-bandwidth version）和“查看低带宽版本”（View low-bandwidth version）。要替换这些用语，你可以将自定义文字传递给forceFailText和forcePassText选项：

```
enhance({
 // 在增强体验里显示：
 forceFailText: "View mobile site",
 // 在基本体验里显示：
 forcePassText: "View enhanced site"
});
```

禁用切换链接

切换链接服务于一个重要的目的，即当某种体验不适合用户时为他们提供一种自愿退出的方法，同时也适用于那些愿意拒绝EnhanceJS结果的用户。既然是为了帮助用户，那么我们永远不会建议你直接删掉它。但是，如果你愿意使用另一种方式来切换基本体验和增强体验，那么EnhanceJS的默认切换链接是可以隐藏的，以防止页面出现多余元素。

为了阻止EnhanceJS插入这个链接，需要在配置enhance函数时将appendToggleLink选项设为false：

```
enhance({
  appendToggleLink: false
});
```

6.3.4　强制通过或不通过EnhanceJS测试

EnhanceJS将通过/未通过能力测试的结果存储在一个cookie里，以供将来载入页面时使用。你可以操纵这个结果，强制网页加载基本体验或增强体验。必须为所有移动设备输出一个单独的“低带宽”版本网站时，就是操作这个结果的一个时机。

EnhanceJS提供了一些辅助方法来简化从一种体验到另一种体验的网页渲染切换。使用这些方法时要小心（而且只在必要时使用），因为它们会绕过能力测试。

- reTest方法会简单地删除cookie并刷新网页，使测试套件重新运行，就好像它之前从未运行过一样。
- forceFail方法将cookie的值设为fail并刷新网页，不应用任何增强信息（即基本体验）。
- forcePass方法将cookie的值设为pass并刷新网页，不管浏览器是否能够正确渲染都会显示增强信息。

要在你自己的脚本里使用这些辅助方法，首先要将通过/未通过的结果储存在一个变量里。这个变量储存enhance函数返回结果的一个引用，可以用它来调用EnhanceJS的辅助方法：

```
//运行默认测试套件
var myTest = enhance();

//强制不通过之前的测试并刷新页面
myTest.forceFail();
```

我们就是在切换链接使用的forceFail和forcePass方法中使用这种方法：EnhanceJS给每个切换链接都绑定一个点击事件，根据当前视图里的体验级别调用适当的方法。

6.4 扩展 EnhanceJS 测试套件

EnhanceJS的默认测试套件会检查一些有代表性的JavaScript对象支持和CSS渲染准确度。但是，你正在开发的项目有可能需要一些我们默认不会测试的技术，例如Flash或者HTML5里的canvas或video等新元素。

我们将EnhanceJS能力测试套件构建得易于定制和扩展。你可以根据需要用多种方式来修改测试套件：利用EnhanceJS内建的配置选项替换或添加套件里的测试项目；创建多个测试实例，面向特定的功能或能力；启用alertonFailure选项来执行大规模的浏览器测试方案。

6

6.4.1 用EnhanceJS选项修改测试套件

EnhanceJS提供了两个用于添加测试的选项：addTests和tests。通过addTests传递来的新测试会添加到现有测试套件中；传递到tests选项里的测试会完全取代默认的测试套件。

这两个选项都接受单个值，用对象格式表示（就是指放在一对大括号里的测试或方法集合）。每种方法都代表一个单项测试，并应该有一个有意义的名称（比如canvasSupport，意思是canvas支持），后接一个冒号和一个普通JavaScript函数，此函数必须返回一个true或false值，这样EnhanceJS才能正确处理结果。这个集合可以列出多项测试，如果的确有多项，它们之间应该用逗号分隔：

```
enhance({
  addTests: {
      canvasSupport: function(){
          return document.createElement('canvas').getContext;
      },
      videoSupport: function(){
          return !!document.createElement('video').play;
      }
  }
});
```

EnhanceJS根据排列顺序依次执行每一项测试，如果它在任何一处遇到一个false值，就会设置一个cookie来表示测试结果为未通过，并且给基本体验添加一个恰当的切换链接；如果所有测试都输出了一个true值，它就会设置一个测试通过的cookie并增强页面。

6.4.2 创建EnhanceJS的新实例或多个实例

EnhanceJS对能力测试采取一种“非全有即全无”（all-or-nothing）的方式：套件里某个单项测试未通过时，测试就停止了，EnhanceJS会记录下一个未通过的成绩。但是，你可能想使用某种功能来增强网页，即使套件里的其他测试未通过，此功能也能在浏览器里成功运行。

你可以在每个必须进行能力测试的网页上创建和运行多个EnhanceJS实例，这让下列操作成为可能。

- 脱离主要能力测试套件，单独测试一项功能（或一组功能）。
- 修改一个现有的EnhanceJS实例，它在某个网站上已经运行了一段时间。EnhanceJS会丢下一个含有通过/未通过结果的永不过期的cookie，为了确保某项功能添加进现有系统时“重置”EnhanceJS，让它忽略cookie并再次运行，就必须创建一个新的测试副本。

每个EnhanceJS实例都必须关联一个独一无二的名称，默认分配的名称是`enhanced`。要创建一个新的EnhanceJS实例，需要调用`enhance`函数，而且要包括一个新的`testName`选项，名称的值则放在引号里。

```
enhance({
  testName: 'enhancedCanvas',
  loadScripts: [
      'js/canvas-enhancements.js'
  ]
});
```

给新实例取名时，要记住EnhanceJS会在整个框架里多次重复使用这个名称，具体会用于：

- 捕捉通过/未通过结果的cookie；
- 添加到切换链接上的前缀（就像`enhanced_toggleResult`）；
- 测试通过时指派给`html`元素的类名。

另外，每一个EnhanceJS实例都会将一个新的体验切换链接添加到页面上，让用户可以切换那个特定功能的基本版本和增强版本。就这一点而言，你应该调整它的样式或位置，使它和那个功能一起出现，这样它就不会和用来切换网页其余部分的链接混淆。举个例子，你可能会单独运行EnhanceJS的一个实例，将某个表格替换成根据它的数据生成的、基于`canvas`的图表。这种独特增强所用的切换链接应该放在图表下面，用“查看表格数据”作为链接文字。

6.4.3　为调试开启能力测试警告

开发和添加你自己的能力测试时，了解哪一项测试导致某个特定浏览器出错是很有帮助的。EnhanceJS提供`alertOnFailure`选项就是为了这个目的。将这个选项设置为`true`后，如果发生测试失败的事件，脚本就会弹出一个JavaScript警告：

```
enhance({
  alertOnFailure: true
});
```

请记住，`alertOnFailure`选项只该用在测试环境里（绝不要用于实际网站），因为它可能会在未通过测试的浏览器里弹出警告，影响用户体验。

为什么EnhanceJS使用`alert()`而非控制台日志

对许多开发者来说，控制台日志工具（比如火狐浏览器的Firebug插件）是在该浏览器上调试脚本时的首选，因为它们的功能更齐全也更健壮。但是，不是所有浏览器都支持控制台调试。

我们发现JavaScript原生的`alert`方法，能在各种各样的新老浏览器里最清楚地显示出在何时何处测试未能通过。它能在几乎所有启用JavaScript的浏览器里触发，而且还支持使用BrowserShots（http://browsershots.org）等有用的工具，此工具能在各种各样的浏览器里测试网页，尤其会显示出测试是在哪个位置失败的。

6.5　在服务器上优化 EnhanceJS

EnhanceJS框架的脚本在能力测试通过时有条件地载入样式和脚本增强信息，这个过程可以得到很大优化，方法是使用服务器端语言（比如PHP、Ruby或Python），根据浏览器上存储的通过/未通过结果输出优化过的网页。

这一过程的工作方式如下：在网页标记里不再编写对enhance.js的引用和调用`enhance`函数的脚本代码块，而是在网页的`head`里放置一个检查EnhanceJS cookie的服务器端包含引用。当用户在他们的浏览器里载入网页时，会发生以下这些事情中的一件。

- 如果cookie不存在（或是因为浏览器之前从未载入过这个网页，或是因为它不支持cookie），就随网页输出所有必需的EnhanceJS脚本，在客户端执行能力测试和应用增强。
- 如果cookie存在且等级是通过，就随网页输出增强信息，包括`html`元素的`enhanced`类、对额外CSS和JavaScript文件的引用，以及体验切换链接。网页到达浏览器时已经准备就绪，所以不必进行客户端的文件载入工作。在这种情况下，服务器端的脚本还提供了一种优化过的方式，将增强标记添加到页面中，之前则需要用客户端JavaScript在测试通过时插入。
- 如果cookie存在且等级是未通过，就以基本形式输出网页，包括基础标记和一个基本样式表，没有任何JavaScript。在这种情况下，体验切换链接仍然会加上，不过不是用JavaScript来切换体验，而是指向一个执行相同功能的服务器端脚本。

注意　要了解EnhanceJS服务器端配置的更多信息和范例代码，请访问EnhanceJS项目的网站：http://enhancejs.googlecode.com。

我们概述了所有需要的基本工具和方法，这样你就能在所有项目上应用渐进增强方法，并有效地进行能力测试以确保普遍可访问性。本书第二部分会逐步介绍12个范例，展示如何应用这些技巧，构建能在真实项目里见到的可访问组件。

Part 2 第二部分

渐进增强实战

本部分内容

第 7 章

用渐进增强方法构建组件

7

用渐进增强方法构建网站和应用程序时，重要的一点是要用X光透视来判断基本体验和增强体验如何协同工作，然后依照可访问性的最佳实践编写标记、CSS和JavaScript，从而向最广泛的浏览器和设备提供尽可能优秀的体验。

不过，首次组合使用这些最佳实践可能令人生畏。所以，这一部分会通过细节丰富的范例一步步带你遍历12个常用的交互组件（例如标签页控件、滑块、数据图表和对话框），并向你展示如何根据我们的渐进增强流程来构建它们。

在深入代码范例之前，我们会花一些时间大致介绍构建这些组件时遵循的大体编程方式，总体介绍涉及的组件，并描述在这部分的各章里你将会见到的组织结构。

7.1 组件是如何编写的

用HTML、CSS和JavaScript标记一张网页有好几种合法方式。接下来几章使用标准的方式和语法，它们的文档丰富且易于理解。

对基础标记而言，我们使用XHTML语法，因为我们发现它的规则严格而明确（比如，所有元素的标签都要关闭，所有属性都要加上引号），因此它易于阅读和复制。

我们使用的元素主要来自最新的HTML正式规范：HTML4.01。我们认为它是“安全”的HTML，因为自1998年4月获得批准后，这部规范在绝大部分移动和桌面浏览器上得到了支持，包括运行在游戏系统和家用游戏机上那些更“创新”的浏览器。同样重要的是，范例里用到的所有元素在下一部HTML规范（HTML5）中会继续得到支持，因此这个标记既是向后兼容的，也是向前兼容的。

写作本书时，HTML5规范还处于草案阶段。虽然它尚未完全获得批准，我们还是在组件范例里使用了许多HTML5的元素和属性，它们可以归为两大类。

- 一些属性（例如输入元素的`type= number`和`data`前缀）能够增加信息，可以用它们实现更加智能的增强体验，给标记整体增加语义价值。同时，它们或者能被不理解它们的浏览器安全忽略，或者有内建的备用HTML4元素。在基础标记里使用它们是安全的。
- 一小组HTML5功能（例如`canvas`和`video`元素）提供了健壮且基于标准的功能性，很可能会加入正式的HTML候选建议稿，而且已经被一些当代浏览器采用。在接下来的组件范例

里，我们建议现在就在你的项目里使用其中的许多功能，我们也会演示怎样做能达到最好的效果。

注意 要了解更多信息，请查阅HTML4.01规范（www.w3.org/TR/html4），HTML4.01指定的元素列表（www.w3.org/TR/REC-html40/index/elements.html），以及HTML4.01和HTML5之间的区别（http://dev.w3.org/html5/html4-differences）。

WAI-ARIA 1.0规范在写作本书时也仍是草案，但已经在一些最为流行的屏幕阅读软件套装里得到了良好支持。我们在基本体验和增强体验的标记里都大量应用了WAI-ARIA可访问性的属性，帮助确保代码能被辅助技术充分理解，并且处在一个良好的位置迎接未来的可访问性进展。

所有组件范例都使用最新的CSS规范：CSS 2.1（www.w3.org/TR/CSS2）里指定的样式属性。只要有可能，我们会加入下一个修订版（通常称为CSS3）提议的那些较新的属性，例如用于圆化边角的`border-radius`或是制造阴影效果的`text-shadow`（www.w3.org/Style/CSS/current-work#CSS3）。目前，很多这些属性只能在一部分现代浏览器里正确渲染，包括最新版的火狐浏览器、Safari以及Opera。在那些不支持它们的浏览器里，它们会被忽略，因此可以放心地将它们用于增强样式，因为我们知道它们没有害处。

7

最后，我们的脚本范例基于jQuery JavaScript库。这个开源库包含许多有用的JavaScript属性和方法，可写出非常简洁却又强大的函数。使用jQuery是因为它的选择器和方法的语法（它在定位和操作具体DOM元素方面的惯例）特别易读，而且能够快速传达特定脚本行为背后的逻辑。jQuery库的文档包含了可接受的选择器和可用方法的完整参考，它在jquery.com上进行维护。（大曝光：Filament集团是jQuery UI组件库的积极贡献者，并作为一个整体加入jQuery领导小组。）

7.2 在组件各章里导航

本书这一部分的每一章都涵盖一个具体的组件范例，只要有可能，我们还会分析该组件的常见变体。组件各章根据其主要目的分为两组。

- 前六章展示了*内容组织组件*（content organization widget），它们排列网页上的信息，使其可以被选择性地显示/隐藏（可折叠内容、标签页、工具提示、对话框和树），或者作为数据可视化方式进行显示（图表）。
- 接下来六章展示了*数据输入和提交组件*（data input and submission widget），它们能提供更便捷（而且经常也是更高效）的方式收集用户输入（按钮、自定义复选框和单选按钮、滑块、列表生成器、下拉菜单和自定义文件输入）。

每一章大致遵循一种格式：用X光透视分析组件的目标设计，将功能部件映射到标准HTML元素上，然后编写基础标记，应用安全样式，叠加上样式和脚本增强信息。

构建每种组件时，我们会遵循一些关键原则，从而确保基本体验和增强体验都能实现恰当的功能，而且让那些使用屏幕阅读器、移动设备和桌面浏览器等方式的用户都能够访问。

- 我们一开始总是将语义化HTML用于基础标记，并且合乎逻辑地定义有效属性。如果目标设计允许，我们会在将这一标记用于基本体验的同时，把它当做一切样式和脚本增强的起点。而且，只要有可能，我们还会将它作为内容来源使用，帮助配置某个增强组件。
- 为了确保每一种增强体验都能像作为其基础的原生标记那样完全可访问，我们加入了ARIA属性、键盘访问能力，以及其他基于现有先例（相似的表单控件、桌面电脑惯例或者WCAG指导方针）的用户期望行为。另外，建议使用VoiceOver、NVDA、WindowEyes和JAWS等屏幕阅读器进行详尽测试。

在数据输入和提交组件方面，我们会特别小心，总是从一个可工作的HTML表单控件起步，确保在没有任何CSS或JavaScript增强时，此组件也能按需工作，并且可由任意设备上的所有用户访问。我们只会添加必要的增强信息，让此元素的外观和行为像一个样式丰富的控件，而不会改变用户与它的交互方式。举个例子，第15章会展示如何使用标准的`input`和`label`元素加上CSS和最少量的JavaScript，定制复选框和单选按钮元素。

如果不能增强某个标准表单控件，我们会确保增强组件（无论它是与原来的表单元素协同工作，还是在用户体验里完全取代后者）总是与用于提交数据的原生输入元素保持不间断的联系。我们通过编写“代理”联系脚本实现这一点：无论何时，只要基本的原生表单控件或者增强的自定义组件改变了数值，这个脚本就会捕捉到用户输入并更新另一个组件，使它们不断保持同步。为了确保与后端的服务器逻辑实现最简单最干净的集成，我们总是会将最初的原生表单元素留在标记里，用来提交表单数据。

运行操作网页元素的脚本增强时，我们会在标记准备就绪后立即开始，这样增强体验就可以无缝地显示出来，还能避免在元素存在前运行脚本所产生的错误。为此，我们使用jQuery的`ready`方法。除非另有说明，可以假定组件各章中所有与DOM有关的代码都在这个`ready`方法内调用。（第5章详细分析了`ready`方法。）

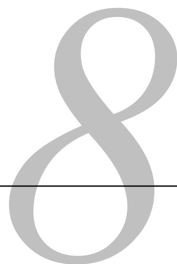
7.3　可下载的范例代码

在接下来的12章里，我们为每一种组件都开发了标记、样式和插件脚本（用于试验和在你的项目里使用）。每一章最后提供了具体的操作说明或提示，方便读者下载与使用组件范例里所描述代码的功能完备版本。本书所有代码都可以在www.filamentgroup.com/dwpe里找到。

所有代码和插件都是开源的，在MIT开源协议用于商业和非商业用途（www.opensource.org/licenses/mit-license.php）。

第 8 章

可折叠内容



网站和Web应用程序的复杂程度一直在增加，设计师们经常会决定选择性隐藏和显示屏幕上的内容，这既可能是根据用户的操作渐进显示，也可能是将内容放在可折叠的块里，让用户可以选择开启和关闭它们。

这种让用户选择性切换其显示状态的内容块有无数种常用设计形式，它们的名字则更多，仅举几例：可折叠的（collapsibles）、转下来的（spin-downs）、切换面板（toggle panels）、卷绕的（twisties）和内容展开（content disclosures）。可折叠内容在许多场合都很有用。

- 某些细节信息不必一直显示或对所有用户显示，但在特定情况下可能有用，比如描述某个技术问题的文档。
- 表单里的可选字段，比如注册表单里的用户偏好设置，或是电子商务结账工作流里的礼品包装选项。
- 仿桌面样式Web应用程序里的工具面板或分组调色面板，用户可以选择折叠或展开某些元素来优化屏幕空间。

在可折叠组件里展示内容时，常用的最佳实践是显示一个视觉指示器（比如箭头或加/减号图标），并附带一个可点击的标签或其他元素，让用户明白有额外的内容可用。

另一个重点是，要确保选择性隐藏的内容对屏幕阅读器仍然是可访问的，方法是添加恰当的ARIA属性，支持键盘访问，以及使用最佳的CSS属性正确管理可视性。本章会讨论如何用可访问的方式构建可折叠内容。

8.1 X 光透视

我们考虑一个经常用到可折叠内容的案例：技术错误的反馈消息。在我们的照片管理器网站里，当用户上传完一批照片后，可能显示一个反馈面板，总结遇到的问题数量，并以简单、紧凑的格式提供具体上传问题的细节，如图8-1所示。

对那些有兴趣了解哪些照片存在问题以及出现了什么具体问题的用户，我们会提供一个“详细信息”（Details）标题，点击后会展开一个面板，里面显示照片问题的完整列表。为了指明这个“详细信息”标题能够显示出更多内容，我们将一个小箭头放在它的左边。面板展开时，这个箭头的方向就变成朝下了，如图8-2所示。

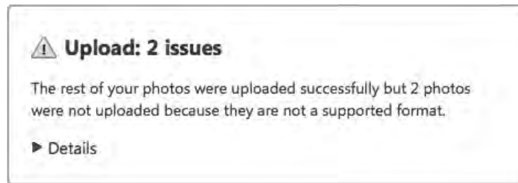


图8-1　细节面板折叠时的目标设计



图8-2　细节面板展开时的目标设计

用X光透视查看这个可折叠详细信息面板时，我们注意到面板里提供的细节内容可能对基本体验里的用户是必要的，能让他们明白接下来该怎么做。因此，要确保它被完整包括在基础标记里。而在增强体验里，默认会用JavaScript和CSS隐藏内容，并添加箭头图标和点击行为，在用户点击“详细信息”标题时展开或折叠内容。

基础标记里有一个要点需要考虑：“详细信息”扮演的是可折叠组件的标题这一角色，因此它应该被标记为一个标题元素（h2），以利用这种标题代表的意义。

除此之外，可折叠面板里的元素可以采用任何语义化HTML的格式。在这个案例里，每一行只包含了两项数据：文件名（我们会把图标做成背景图像）和问题描述，格式也非常简单，因此可以使用一张无序列表使标记轻量化。把问题描述放在一个强调标签（em）里，就可以用CSS使它浮动到右侧，从而符合我们的设计，如图8-3所示。（如果细节数据包含了任何额外的数据字段，或者添加解释性的表头会更好，那么我们很可能将它标记为一张表格。）

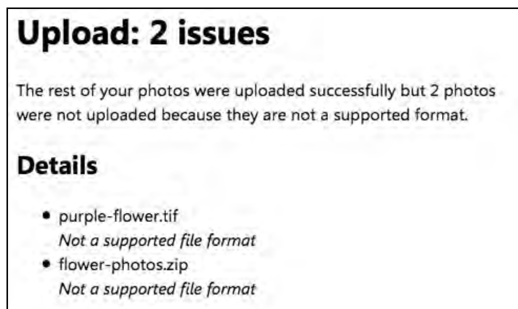


图8-3　基本体验显示所有可折叠的内容，因此是可访问的

增强版标记和脚本必须包括一些重要的可访问性功能，以确保它遵循为屏幕阅读器用户服务的最佳实践标准。

使用屏幕阅读器的用户在页面朗读时希望：阅读展开的内容，跳过折叠的内容。为了避免朗读折叠的内容，我们会在增强版CSS里用`display: none`隐藏它，并在DOM里给它加上`aria-hidden="true"`标记。

这种隐藏折叠内容的方式是可行的，前提是告知屏幕阅读器用户该内容的存在，并提供一种访问它的方法。增强脚本会对“详细信息”标题做一点必要的标记改动。

- 此标题需要一些提示，让用户知道它包含更多内容。为了提供明确的操作说明，脚本会在标题文字前插入一个span，检测可折叠面板当前的状态，在内容折叠时加入文本“显示”（Show），展开时则加入文本“隐藏”（Hide。请注意文本后面的空格，它让屏幕阅读器暂停一下再朗读标题文字）。屏幕阅读器会读出更具描述性的“显示详细信息”或“隐藏详细信息”，解释用户点击后会发生什么。这些span元素会用可访问的方式隐藏到屏幕之外，这样在网页上就看不见它们了。
- 此标题必须能够获得键盘焦点。脚本会用一个锚元素（a）围住标题标签里的文字，这样当用户按下Tab键时就能导航到它上面。

我们已经充分了解了如何构建基本体验和增强体验，因此可以着手编写基础标记和样式了。

8.2　创建可访问的可折叠内容

创建在基本体验和（用屏幕阅读器访问的）增强体验里完全可访问的可折叠内容组件，是一个相当简单的过程。

在接下来的代码攻略里，我们会特别关注可点击标题和目标设计里的可折叠部分，如图8-4所示。

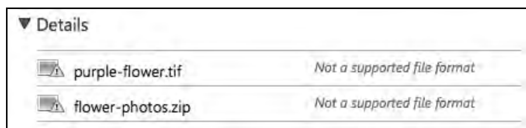


图8-4　目标设计里的可折叠部分

8.2.1　基础标记和样式

可折叠面板的基础标记很简单，就是一个标题后面跟着一张无序列表。在每一个列表项里，问题描述被放在一个强调标签内，和文件名区分开：

```
<h2>Details</h2>
<ul>
  <li>purple-flower.tif <em>Not a supported file format</em></li>
  <li>flower-photos.zip <em>Not a supported file format</em></li>
</ul>
```

在基本样式表里，我们会在body标签上设置字体，并给每个em元素添加display: block样式，让问题描述单独占一行：

```
body { font-family: "Segoe UI", Arial, sans-serif; }
ul li em { display: block; }
```

在基本体验里，所有内容在任何时候都可见，没有折叠行为。应用安全样式后它变得易于阅读，并被组织成一种清晰的视觉层级，如图8-5所示。

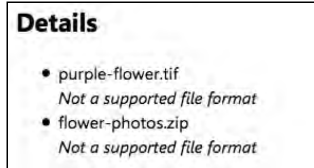


图8-5　应用安全样式后的基本体验

8.2.2　增强标记和样式

要在增强体验里给展开和折叠状态添加样式，我们需要确定脚本将要应用到标记上的类，用来添加样式及启用显示和隐藏行为。

“详细信息”标题被指派了一个collapsible-heading类，内容里的ul则被指派了collapsible-content类。这些类用于定义展开状态的外观：将标题箭头图标的方向指向下方，以及使细节内容在屏幕上可见：

```
<h2 class="collapsible-heading">Details</h2>
<ul class="collapsible-content">
  <li>purple-flower.tif <em>Not a supported file format</em></li>
  <li>flower-photos.zip <em>Not a supported file format</em></li>
</ul>
```

用户点击标题来隐藏细节内容时，脚本会给标题添加一个collapsible-heading-collapsed类，以将箭头图标换成指向右方，之后给列表添加一个collapsible-content-collapsed类，让它在屏幕上隐藏起来，并且对大多数屏幕阅读器也隐藏起来。脚本还会更新aria-hidden属性进行匹配：

```
<h2 class="collapsible-heading collapsible-heading-collapsed">Details</h2>
<ul class="collapsible-content collapsible-content-collapsed" aria-hidden=
"true">
  <li>purple-flower.tif <em>Not a supported file format</em></li>
  <li>flower-photos.zip <em>Not a supported file format</em></li>
</ul>
```

X光部分提到，这个标题需要额外的标记使它可以用键盘导航访问，以及为屏幕阅读器提供上下文环境的操作说明。

首先，我们用一个带有collapsible-heading-toggle类的链接围住标题文字，这样它就可以

被键盘用户访问到。添加一个名为button的ARIA角色，则可以在能理解新型ARIA标记的浏览器里标明此元素的作用：

```
<h2 class="collapsible-heading">
  <a class="collapsible-heading-toggle" href="#" role="button">Details</a>
</h2>
```

脚本还会在文字前插入一个带有collapsible-heading-status类的span，我们用它为屏幕阅读器用户提供操作说明。脚本还会包含一套逻辑，根据可折叠面板的状态输出恰当的span文字。

```
<h2 class="collapsible-heading">
  <a class="collapsible-heading-toggle" href="#" role="button">
  <span class="collapsible-heading-status">Hide </span>
  Details</a>
</h2>
```

当内容展开以及链接获得焦点时，屏幕阅读器会将此链接读作“隐藏详细信息”：

```
<h2 class="collapsible-heading collapsible-heading-collapsed">
  <a class="collapsible-heading-toggle" href="#" role="button">
  <span class="collapsible-heading-status">Show </span>
  Details</a>
</h2>
```


确定标记和类名之后，我们可以为其编写增强样式规则，将必要的样式应用到组件上。

首先，我们会在增强样式表里设置一个全局字体大小，这样做会简化我们设置组件内各元素字体大小的工作。我们会赋予body元素一个font-size属性，以“重置”标准的浏览器默认字体大小。

我们在设置字体大小时喜欢使用相对式的em单位，这样用户通过浏览器的“首选参数”设置和键盘命令，能更容易地把文字调整到自己喜欢的大小。大多数浏览器的基准字体大小是16像素。通过将body字体大小设置成16像素的62.5%，我们实际上将单个em单位（1em）“重置”为10像素：

```
body { font-size: 62.5% }
```

因为基准字体大小被设置成百分比的形式，所以任何指定的em都会简单地转换成倍数（1.5 em等于15 px，2.2 em则是22 px，以此类推），这会大大减少计算每种元素文字大小的需要，还能够保持可伸缩性，适合那些修改了默认大小或者使用键盘命令（例如Ctrl+或Ctrl-）即时调整大小的用户。

致　　谢

这种基于百分比的字体大小解决方案灵感来源于Richard Rutter的文章。他在2004年5月撰写的博客文章“How to size text using ems”（www.clagnut.com/blog/348）中列出了原则和代码范例，并对在网页中嵌套em单位时如何处理倍增效应给出了非常有帮助的解释。

接下来，我们会为各个类编写样式规则，但不具体引用那些应用它们的元素。利用这种一般性方法，我们的可折叠行为能在不同的HTML元素组合中正确工作，比如一个h3后面跟着一个p。

collapsible-heading需要15像素的左内边距，给箭头背景图像留出空间：

```
.collapsible-heading {
  padding-left: 15px;
  background: url(../images/icon-triangle.png) 0 6px no-repeat;
}
```

箭头背景图像是一个组合图像（sprite），包含展开（朝下）和折叠（朝右）两种状态。如果当前是折叠状态，背景位置就变成显示朝右箭头：

```
.collapsible-heading-collapsed {
  background-position: 0 -84px;
}
```

我们希望标题能够匹配目标设计，这就意味着我们不想让h2继承切换链接的默认下划线，因此会去掉它：

```
.collapsible-heading-toggle {
  text-decoration: none;
}
```

（这一步是可选的。如果你确实希望可折叠容器的标题保留原生的下划线，使它看起来像个链接，那么只需跳过这一步。）

当无序列表内容包含折叠状态类时，我们将display属性设置成none，使它同时在屏幕上和对屏幕阅读器隐藏：

```
.collapsible-content-collapsed {
  display: none;
}
```

为了让标题（“隐藏”或“显示”）前面的span处在能够被屏幕阅读器阅读的位置（但在屏幕上是隐藏的），我们把它放到屏幕之外远远的地方：

```
.collapsible-heading-status {
  position: absolute;
  left: -99999px;
}
```

最后，我们会给可折叠组件内容应用格式和样式，让它看上去更接近目标设计。这一部分的视觉设计和两个特定元素（h2、ul）有关，因此我们会在选择器里指定它们：

```
h2.collapsible-heading {
  font-size: 1.5em;
  font-weight: normal;
}

h2.collapsible-heading .collapsible-heading-toggle {
  color: #333;
}

ul.collapsible-content {
  border-top: 1px solid #aaa;
  padding-left: 0;
```

```
  margin-left: 16px;
}

ul.collapsible-content li {
  line-height: 1;
  font-size: 1.3em;
  color: #000;
  border-bottom: 1px solid #aaa;
  padding: .5em 0;
  list-style: inside url(../images/icon-file-warning.png);
}

ul.collapsible-content li em {
  color: #666;
  font-style: italic;
  float: right;
  margin-right: 40px;
  font-size: .9em;
}
```

至此，我们就有了基础标记结构和样式，用来实现完全可访问的组件展开和折叠状态，如图8-6所示。

8



图8-6　组件的折叠状态（顶部）和展开状态（底部）

8.2.3　实现可折叠的增强脚本

增强脚本会将基础标记转变成增强组件，方法是自动添加新的标记和类，并将点击行为关联到标题上来切换状态类。

1. 生成增强标记

首先，创建对标题和内容元素的变量引用，这样就可以用脚本操作它们：

```
//用jQuery找到h2
var collapsibleHeading = $('h2');

//jQuery的next()方法找到下一个同级元素
var collapsibleContent = collapsibleHeading.next();
```

我们会给标题附上collapsible-heading类，添加状态span和显示/隐藏文字，并用链接标签围住标题文字：

```
  collapsibleHeading
    .addClass('collapsible-heading')
```

```
    .prepend('<span class="collapsible-heading-status"></span>')
    .wrapInner('<a href="#"
    ➥class="collapsible-heading-toggle" role="button"></a>')
```

赋予内容列表一个collapsibleContent类：

```
collapsibleContent
  .addClass('collapsible-content');
```

2. 给增强标记应用行为

接下来，脚本会把事件指派给标题，从而展开和折叠内容。这里会使用自定义事件，这样就可以在鼠标点击标题时利用程序触发展开和折叠行为。

首先，我们会创建一个自定义折叠事件，它将折叠类指派给标题，修改状态文字为"显示"，并将折叠类和aria-hidden属性添加到内容里：

```
collapsibleHeading.bind('collapse', function(){
  //在这个上下文中， $(this)指的是标题
  $(this)
     //添加折叠类，让箭头图标朝右
     .addClass('collapsible-heading-collapsed')
     //把可访问环境里的span改成"显示"
     .find('.collapsible-heading-status').text('Show ');
  collapsibleContent
      //将折叠类添加到内容容器
      .addClass('collapsible-content-collapsed')
      //将aria-hidden属性设置为true
      .attr('aria-hidden',true);
 });
```

我们还会创建一个自定义事件来展开内容，大致过程就是折叠过程的反转（移除和切换属性，修改文字以反映展开状态）：

```
collapsibleHeading.bind('expand', function(){
  //在这个上下文中，$(this)指的是标题
  $(this)
      //移除折叠类，让箭头图标朝下
      .removeClass('collapsible-heading-collapsed')
      //把可访问环境里的span文字改成"隐藏"
      find('.collapsible-heading-status').text('Hide ');

 collapsibleContent
     //移除内容容器中的折叠类
     .removeClass('collapsible-content-collapsed')
     //将aria-hidden属性设置为false
     .attr('aria-hidden',false);
   });
```

3. 触发自定义展开/折叠事件

我们已经创建了自定义事件，因此只需在恰当的时候触发它们就行了。增强脚本会在网页载入时立即触发折叠行为，让内容在默认情况下自动折叠：

```
collapsibleHeading.trigger('collapse');
```

我们还会编写逻辑代码，在用户点击标题时触发恰当的事件（如果内容是折叠的就展开，如果内容是展开的就折叠）。可以使用一条简单的if/else语句做到这一点。点击发生时，我们会检查标题上是否存在折叠类，然后触发恰当的事件。我们会对点击事件返回false作为结束，以确保原生链接行为不会跟着出现（如果出现了，那么一个#字符就会附加到URL尾部，而大多数浏览器会滚动到页面顶部）：

```
collapsibleHeading.click(function(){
  //如果标题有折叠类就展开它
  if( $(this).is('.collapsible-heading-collapsed') ){
      $(this).trigger('expand');
  }
  //否则就折叠它
  else {
      $(this).trigger('collapse');
  }
  //返回false，以避免默认的锚点击
  return false;
 });
```

现在，我们就有了一个功能完整的可折叠内容组件，对键盘和屏幕阅读器都是完全可访问的。

8.3 使用可折叠脚本

随书附带的演示和可下载代码（位于www.filamentgroup.com/dwpe）包含一个名为jQuery.collapsible.js的脚本。它在一个简单可复用的函数里实现了这一章所有的原则，你可以将这个函数应用到任何网页元素上。

要使用这个脚本，请下载并引用可折叠演示页里列出的那些文件。要让网页里的某个标题表现得像一个可折叠组件，只需指定jQuery选择器，对其简单地调用collapsible方法：

```
$('h2').collapsible();
```

在这里例子里，可折叠脚本会将可折叠行为应用到网页里的每个h2上。不需要指定此插件应该展开和折叠哪个内容块，因为它假定下一个元素就是内容，这个元素紧跟在对其调用collapsible方法的标记之后（在我们的范例里是无序列表）。

给每个h2都应用这一行为，对真实世界里的应用程序而言可能有些范围过大，所以可以使用jQuery的选择器找到更小范围内的可折叠元素，可以是ID或类属性，也可以是像下面这样的派生选择器（descendent selector），它会将可折叠行为应用到某个特定div里的所有h2元素上：

```
$('#main h2').collapsible();
```

我们在许多增强体验里经常使用这一章描述的技巧来显示和隐藏内容(包括在视觉上和对屏幕阅读器)，它们能让我们响应式地将内容和功能面向合适的受众。

这些基本的概念和原则（设计类，应用键盘访问，添加有用的操作说明语句，以及隐藏和显示）可应用到网页的任何内容上。它们可以用于构建更为复杂的组件，或者创建出能根据用户的选择，以可访问的交互方式隐藏和显示表单元素的表单。

第 9 章

标　签　页

9

标签页（Tabs）将内容整齐地组织成一组逻辑版块，通过每次只显示一个版块节省了界面空间。标签页一下子就能看出来并且符合直觉，因为它模拟了真实世界范例的外观和行为，比如标签式的马尼拉文件夹（manila file folder），或者某本通讯录里的标签分类区域。

在网站或应用程序界面里使用标签页的方式有很多，包括下面这些。

- 展示一个紧凑的内容模块，例如某个新闻网站上“最多邮件发送/最流行/最多讨论”的一组文章，或是像幻灯片那样循环播放的一组功能。
- 作为局部导航或主要视图切换工具，每个标签都会置换掉网页的一大块区域，比如某个出版应用程序里的编辑和预览模式。
- 将与某个单选按钮元素相关联的一组表单元素在视觉上进行归组，比如关于X光透视的第2章里的那个结账支付范例。

标签组的首要功能是选择隐藏和显示某些可见内容。要点是要确保每个内容块的隐藏方式仍然能被屏幕阅读器访问。为了进一步提升可访问性，关键在于使用ARIA属性并添加脚本，确保屏幕阅读器用户既能进行导航，又能用键盘选择标签页。

这一章会向你展示如何制作一组标签页，让基本体验里的内容既有组织又可用，以及如何将基础标记转换成可访问的标签页，让所有人都满意并且可以使用。

9.1　X 光透视

假设我们正在构建一个新闻网站，它的首页内嵌一个用于显示精选内容的小块，显示一些标签式的版块：突发新闻、热门体育新闻以及未来5天的天气预报，如图9-1所示。

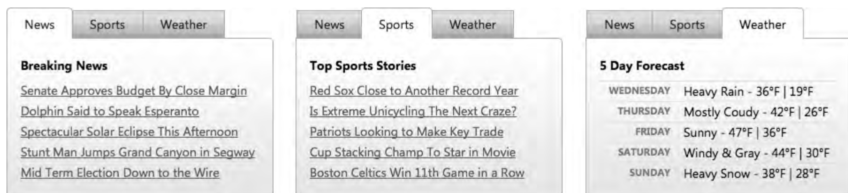


图9-1　增强体验的目标新闻设计

有两种方式可以编写具备语义意义的标签页组件基础标记。

▶ 将三个标签页版块写成连续的块，每个块都有一个标题，后面紧跟着它的内容，并将标题打造成标签页的样式。

▶ 创建一张由锚链接构成的无序列表，对应相关的内容块。用户点击某个链接时，网页会滚动到具体的内容块上。

这两种标记选择在基本体验里都是成立和可用的。就我们这个新闻范例而言，各个内容块都包含了描述性的头信息，可以将它们作为增强体验里的标签页标题。这种方法的效果非常好，而且实施起来也非常简单。

但对于紧凑的精选块里为标签预留的可用空间来说，这些头信息里的文字有些长。而且在移动设备上，这些堆叠起来的内容块可能会变得很长，被推向网页下方，这就意味着用户无法立即看到和使用第二个块和第三个块。

用一列锚链接实现标签页，我们可以使用短名称来指代这三个标签页版块。另外，它们还是一种紧凑的跳跃导航方式，在网页的最顶部显示所有选项。使用锚链接还有一个好处：用户在基本体验里可以收藏某个特定的内容块，因为锚的`href`值（即#号串）会被自动添加到浏览器地址栏（比如`index.htm#sports`），这个功能用静态标题元素无法实现。最后，锚链接和它关联的内容不必非得在标记里按顺序排列，可以用HTML语义手段将它们联系到一起，方法是让每个链接的`href`值（`href="#sports"`）与在特定内容块上设置的ID（`div id="sports"`）相匹配。

出于这些原因（紧凑性、易于跳跃导航、添加书签以及可以随意组织网页结构的灵活性），我们将使用锚链接构建标签页。

构建完这种标记结构后，基本体验在缺少CSS和JavaScript的情况下结构清晰而且可用。这个基础标记结构还天生可以用键盘访问，并且在任何屏幕阅读器上应该都能工作，因为它使用了语义化HTML，如图9-2所示。



图9-2　基本体验里的标签页内容

我们会为那些有能力渲染出增强体验的浏览器叠加上额外的样式和行为,将这些结构化内容转变成一个标签页组件，并添加ARIA属性和键盘行为，确保屏幕阅读器的可访问性。

ARIA规范为屏幕阅读器提供了一组详细的属性，用于恰当地描述某个标签页组件里各个子组件的角色和状态，我们的增强脚本会将它们应用到增强标记上。ARIA还为在标签页组件里用键盘导航推荐了特定的交互行为。具体来说，它建议标签条里的各个链接（“新闻”News、“体育”Sports、“天气”Weather）要统一表现为单个UI控件。当某个标签条获得焦点时，用方向键应该能前后切换标签页。当用户的焦点在某个标签上时，按下Tab键应该会使焦点离开标签条，跳到网页里下一个可获得焦点的元素上：如果当前打开的标签页里有可获得焦点的链接或者表单元素，就会跳到这块内容上；如果没有，则会跳到标签页组件之外的某个元素上。这一行为需要用脚本实现，因为Tab键在正常情况下会将焦点移过标签条里的每个链接，而方向键的原生作用是滚动浏览器的窗口。

我们还得决定在增强体验里如何处理URL#号串。之前提到，在基本体验里，对书签和后退按钮的支持原生内建于内部锚链接（意指这些链接引用的是同一张网页里的内容）。当用户点击某个锚链接时，网页的URL（index.htm）会自动更新，加上锚链接的“#号串”目标（index.htm#sports），之后它就可以被收藏了。点击后退按钮后，网页会返回之前的#号串以及相应的滚动位置上。

如果标签页组件控制了网页里非常大的一块区域,用户也许会将标签视为网页级别的导航元素，可能会期望能收藏特定的标签页视图或使用后退按钮来切换它们。为此，必须添加额外的JavaScript逻辑，在用户点击标签链接时用程序更新URL#号串，还要查看URL里是否存在（或改变了）#号串，依此显示出适当的标签页内容来匹配当前的#号串值。

然而，在这个特定的目标设计案例里，是非常小的标签页组件内嵌于一张更大的网页之中。如果在它内部进行的交互能够影响浏览器历史和后退按钮的行为，就可能令人感觉困惑和不便。举个例子，如果用户先访问体育标签页，再去天气标签版块，他们就需要点击后退按钮三次才能回到上一张网页（index.htm #weather→index.htm#sports→index.htm→上一个页面）。因此，我们不想在用户每次点击标签时都更新URL#号串。对这样的小型标签页组件，可以用JavaScript来禁止锚链接默认的更新URL#号串行为。

我们首先介绍如何创建一个简单的新闻标签页组件（从构建它的基础标记到添加增强信息），然后再概述如何扩展某个标签页组件，使它支持书签和历史追踪（后退按钮）功能。

9.2　创建标签页

为了构建新闻标签页组件，首先我们会从一列锚链接开始编写结构良好的基础标记，将内容组织得尽可能清晰，然后给有能力的浏览器应用样式和脚本增强信息。

9.2.1　基础标记和样式

我们会将标签页标记为无序列表里的一组锚链接，这些链接通过引用ID指向一个具体的内容

块。在基本体验里，点击一个链接会将网页滚动到它关联的内容块上，并用合适的#号串更新URL（例如index.htm#news）：

```
<ul>
  <li><a href="#news">News</a></li>
  <li><a href="#sports">Sports</a></li>
  <li><a href="#weather">Weather</a></li>
</ul>
```

在列表下方，各个标签页内容块都被编写成一个div，它带有一个唯一的ID，关联上方的锚链接：

```
<div id="news">...</div>
<div id="sports">...</div>
<div id="weather">...</div>
```

在前两个标签页（新闻和体育）里，我们会使用一个h2作为标题，后面跟着一张内含文章链接的无序列表。在基本体验里，这些链接会显示成易于阅读的带符号（bulleted）列表格式：

```
<div id="news">
<h2>Breaking News</h2>
  <ul>
      <li><a href="senate.htm">Senate Approves Budget By Close
      Margin</a></li>
      ...
  </ul>
</div>
```

天气内容块同样也会使用一个h2作为标题，然后使用一张定义列表（即dl）来展现一周内的各天以及相应的天气预报。在基本体验里，大多数浏览器默认会给日期（dt）和天气预报（dd）渲染出清晰的区别：

```
<div id="weather">
  <h2>5 Day Forecast</h2>
  <dl>
      <dt>Wednesday</dt>
      <dd>Heavy Rain - 36&#176;F | 19&#176;F</dd>
      ...
  </dl>
</div>
```

一个ID为featured的容器div会围住整个标签页结构（链接列表和内容块），以描述内容的用途。它内部的无序列表会被指派一个featured-links的ID，所有的内容块（新闻、体育和天气）会被归组进一个ID为featured-content的容器div里。虽然我们主要会在增强体验里使用这些容器来给标签页组件添加样式，但它们也能帮助我们在基本体验里归组和组织内容：

```
<div id="featured">
  <ul id="featured-links">
      <li><a href="#news">News</a></li>
      <li><a href="#sports">Sports</a></li>
      <li><a hrer="#weather">Weather</a></li>
  </ul>
 <div id="featured-content">
     <div id="news">...</div>
```

9

```
    <div id="sports">...</div>
    <div id="weather">...<div>
 </div>
</div>
```

不需要多少CSS来装饰这个基本体验，因为我们用的是非常有描述性的标记（例如标题、列表和锚链接），它们会赋予内容一种清晰、可用的格式。为了保持简洁，我们只在基本样式表里加入一条规则，设置基本字体偏好，其他的样式则交给浏览器用默认值处理：

```
body { font-family: "Segoe UI", Arial, sans-serif; }
```

现在，我们就有了创建基本体验所需的、应用了安全样式的基础标记（如图9-2所示）。

9.2.2　增强标记和样式

标签页增强脚本运行时，它会给标记加上许多的类和ID来应用标签页组件样式和脚本行为。

我们会将tabs-nav类分配给由锚链接组成的无序列表，并给每个列表项指派一个由脚本生成的ID，它由锚ID和tab-前缀组合而成。我们还会将tab-selected类分配给当前选中的标签，稍后应用样式规则，使它高亮显示并在视觉上与该标签的内容容器相连。

我们会将tabs-body类添加到外侧的标签页内容容器上，并给它里面的每一个标签页内容div指派一个tab-panel类。当前选中的标签面板也会分配到一个tabs-panel-selected类，用来切换标签页内容的可见性：

```
<div id="featured">
  <ul id="featured-links" class="tabs-nav">
      <li class="tab-selected" id="tab-news"><a href="#news">News</a></li>
      <li id="tab-sports"><a href="#sports">Sports</a></li>
      <li id="tab-weather"><a href="#weather">Weather</a></li>
  </ul>
  <div id="featured-content" class="tabs-body">
      <div id="news" class="tabs-panel tabs-panel-selected">
         ...
      </div>
      <div id="sports" class="tabs-panel">
         ...
      </div>
      <div id="weather" class="tabs-panel">
         ...
      </div>
  </div>
</div>
```

至于增强体验的可访问性支持，我们会添加若干ARIA和tabindex属性。首先，给body元素应用application这个ARIA role，指示屏幕阅读器将其视为一个应用程序式样的组件，而不是标准网页内容：

```
<body role="application">
```

我们会将tablist这个role指派给锚链接，每个列表项的role则是tab。它们的作用是通知屏

幕阅读器用户这些链接组成了可点击的标签条。各个内容div都会获得一个ARIA role="tabpanel"，标明它是某个标签的内容块，也会获得aria-labelledby属性，将它及其对应标签的id关联起来。另一个属性aria-hidden（设置为false）标明它是当前可见的标签面板，我们会在其他的面板上设置true值。另外，我们会给每个标签应用一个tabindex属性，活动标签的值是0，其他的则是-1：

```
<div id="featured">
  <ul id="featured-links" role="tablist" class="tabs-nav">
      <li role="tab" class="tabs-selected" id="tab-news">
      <a href="#news" tabindex="0">News</a></li>
      <li role="tab" id="tab-sports"><a href="#sports"
      tabindex="-1">Sports</a></li>
      <li role="tab" id="tab-weather"><a href="#weather"
      tabindex="-1">Weather</a></li>
  </ul>
  <div id="featured-content" class="tabs-body">
      <div id="news" role="tabpanel" aria-labelledby="tab-news"
      aria-hidden="false" class="tabs-panel tabs-panel-selected">
      ...
      </div>
      <div id="sports" role="tabpanel" aria-labelledby="tab-sports"
      aria-hidden="true" class="tabs-panel">
      ...
      </div>
      <div id="weather" role="tabpanel" aria-labelledby="tab-weather"
      aria-hidden="true" class="tabs-panel">
      ...
      </div>
  </div>
</div>
```

通过这种方式把tabindex属性应用到锚链接上，就能覆盖它们的原生行为，把它们从独立可获得焦点的元素转变成只有一个元素能获得焦点的单个组件。随着焦点在标签之间转移，我们会用JavaScript把tabindex="0"这个值（可获得焦点）只放到当前选中的标签上，其他则统一设置成tabindex="-1"。动态改变这个值能确保每次这个组件在网页上获得Tab焦点时，焦点会回到活动标签上。这一技巧被称为*游动标签序号*（roving tab index）。

完成标记后，就可以接着给标签页添加样式来匹配我们的目标设计。在增强样式表里，首先会给用于标签条的无序列表添加样式：将它下移1像素重叠到面板上，移除列表项前面的符号，浮动各个列表项让它们并排显示，并给链接添上背景图像和边框样式，效果如图9-3所示。

```
.tabs-nav {
  height:2em;
  margin:0;
  padding:0;
  padding-left:1px;
  bottom:-1px;
  list-style:none;
  position:relative;
}
```

```
.tabs-nav li {
  float:left;
  margin:0;
  padding:0;
}
.tabs-nav li a {
  float:left;
  padding:.3em 1.4em .4em;
  text-decoration:none;
  font-size:1.4em;
  border:1px solid #aaa;
  border-bottom:0;
  background:#ddd url(../images/bg-tab.png) 50% 50% repeat-x;
  margin-right:-1px;
  color:#222;
}
```

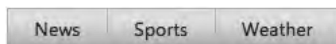


图9-3　应用了标签条样式的链接列表

接下来，编写一些CSS规则，改变鼠标悬停时的标签外观：

```
.tabs-nav li a:hover {
  background:#eee url(../images/bg-tab-hover.png) 50% 50% repeat-x;
  color:#000;
}
```

然后，给选中状态应用样式。选中的标签会在底部显示白色边框，在视觉上和标签面板及圆化的顶角连起来。我们还会添加CSS3的`border-radius`属性来圆化选中标签的两个顶角（别担心，不支持这个属性的浏览器只会显示方形的边角）。默认情况下，`border-radius`会应用到对象的全部四个边角上。要圆化某个特定的边角，每一个目标边角都需要设置三个属性以适应各种浏览器的实现方式：举个例子，如果只想圆化右上角，那么需要设置标准的`border-top-right-radius`属性，以及两个`-moz`和`-webkit`版的浏览器专用属性，效果如图9-4所示。

```
.tabs-nav li.tabs-selected a {
  position:relative;
  background:#fff;
  padding-top:.5em;
  margin-top:-.2em;
  border-bottom:1px solid #fff;
  -moz-border-radius-topright:5px;
  -webkit-border-top-right-radius:5px;
  border-top-right-radius:5px;
  -moz-border-radius-topleft:5px;
  -webkit-border-top-left-radius:5px;
  border-top-left-radius:5px;
}
.tabs-nav li.tabs-selected {
  margin-left:-1px;
  }
```



图9-4 标签交互状态（从左到右）：已选中、鼠标悬停和默认状态

我们给包含标签内容面板的容器（tabs-body）设置一个固定的宽度，并给它一个边框和背景图像来增加一点界定感和质感。除左上角之外（我们让它保持方形，与上面的标签相连），它的所有边角都被圆化了。每个标签内容div（tabs-panel）都设置成了display:none，在默认情况下是隐藏的。某个标签被选中时，脚本会将tabs-panel-selected类添加到关联的标签内容面板上，从而将它设置成display: block，使其可见，效果如图9-5所示。

```
.tabs-body {
  clear:both;
  overflow:auto;
  border:1px solid #aaa;
  width:300px;
  background:#ddd url(../images/bg-tab-body.png) 50% bottom repeat-x;
  -moz-border-radius-topright:5px;
  -webkit-border-top-right-radius:5px;
  border-top-right-radius:5px;
  -moz-border-radius-bottomright:5px;
  -webkit-border-bottom-right-radius:5px;
  border-bottom-right-radius:5px;
  -moz-border-radius-bottomleft:5px;
  -webkit-border-bottom-left-radius:5px;
  border-bottom-left-radius:5px;
}
.tabs-body div.tabs-panel {
  padding:15px;
  overflow:auto;
  display:none;
  font-size:1.4em;
}
.tabs-body div.tabs-panel-selected {
  display:block;
}
```



图9-5 完成后的增强体验标签样式

9.2.3 标签页脚本

完成标记草案和样式后，现在可以编写将基础标记转变成标签页组件的增强脚本，用在通过能力测试的那些浏览器上。

首先，定义一些变量来引用标签页组件里那些重要的元素，包括外部容器、锚链接列表和标签内容的容器。那些用来生成唯一ID的ID前缀（tabs-）和ARIA应用程序模式都会在这里预先定义：

```
//引用标签页容器
var tabs = $('div#featured);

//设置应用程序模式
$('body').attr('role','application');

//nav是第一个ul
var tabsNav = tabs.find('ul:first');

//body是nav的下一个同级元素
var tabsBody = tabsNav.next();

//用于创建标签ID的前缀，创建依据的是对应div的ID
var tabIDprefix = 'tab-';
```

接下来，将所有类、ID、ARIA和tabindex属性添加到基础标记，以匹配增强标记：

```
//为nav添加类和aria
tabsNav
  .addClass('tabs-nav')
  .attr('role','tablist');

//为标签页body添加类
tabsBody.addClass('tabs-body');

//找到标签面板，添加类和aria
tabsBody.find('>div').each(function(){
  $(this)
      .addClass('tabs-panel')
      .attr('role','tabpanel')
      .attr('aria-hidden', true)
      .attr('aria-labelledby', tabIDprefix + $(this).attr('id'));
 });

//设置每个标签的角色
tabsNav.find('li').each(function(){
  $(this)
  .attr('role','tab')
  .attr('id',
  tabIDprefix +$(this).find('a').attr('href').split('#')[1] );
});

//将所有标签的tabindex设为-1
tabsNav.find('a').attr('tabindex','-1');
```

用户选择一个标签时，我们会添加和移除一些类，用选中状态来高亮显示正确的标签，调整各个标签的tabindex值，把焦点设定到选中的标签上，并显示出关联的标签内容块。我们需要一种可以重复使用的通用程序方式来选择标签，因为它们会以多种方式被选中（鼠标点击、键盘事件，以及网页首次载入时）：

```
//选择某个标签的通用函数
function selectTab(tab){
 //取消选中活动标签
 tabsNav
     .find('li.tabs-selected')
         .removeClass('tabs-selected')
         .find('a')
             .attr('tabindex','-1');
 //选择新的选项卡
 tab
     .attr('tabindex','0')
     .parent()
         .addClass('tabs-selected');
 //取消选中活动面板
 tabsBody
     .find('div.tabs-panel-selected')
         .attr('aria-hidden',true)
         .removeClass('tabs-panel-selected');

 //选中新面板
 $( tab.attr('href') )
     .addClass('tabs-panel-selected')
     .attr('aria-hidden',false);
 //如果没有其他命令，就更新URL#号串并让标签获得焦点
 window.location.hash = tab.attr('href').replace('#','');

 //将焦点转移到新选中的标签上
 tab[0].focus();
};
```

有了这段脚本，我们就可以用想要选中的标签锚元素作为参数来调用selectTab函数。现在，我们已经准备好应用鼠标、键盘和其他事件来将它们绑定到一起了。

首先，我们会给标签链接应用事件。用户用鼠标点击某个标签时，我们会将那个标签传递给通用的selectTab函数。绑定这个事件时，我们会return false以避免出现原生的锚链接行为（就是指更新URL#号串，并让焦点跳离标签链接，转移到内容块上）：

```
tabsNav.find('a')
  .click(function(){
     selectTab( $(this) );
     return false;
  });
```

下一个需要考虑的交互点是键盘。我们会追踪标签拥有焦点时触发的任何keydown事件，并映射特定的键用于将焦点转移到另一个标签上：左方向键和上方向键会选中上一个标签，而右方向键和下方向键会选中下一个标签。开始键（Home）和结束键（End）则分别选择最前和最后的

标签，我们会以绑定逻辑代码到特定键码（key code，比如36就是开始键）的方式关联键盘事件：

```
tabsNav.find('a')
  .keydown(function(event){
      var currentTab = $(this).parent();
      switch(event.keyCode){
          case 37: //左方向键
          case 38: // 上方向键
              if(currentTab.prev().size() > 0){
              selectTab( currentTab.prev().find('a'));
              }
              break;
          case 39: // 右方向键
          case 40: // 下方向键
              if(currentTab.next().size() > 0){
              selectTab( currentTab.next().find('a') );
              }
              break;
          case 36: // 开始键
              selectTab( tabsNav.find('li:first a') );
              break;
          case 35: // 结束键
              selectTab( tabsNav.find('li:last a') );
              break;
          }
 });
```

最后，我们会在网页载入时打开第一个标签面板。通过把第一个标签的引用传递给selectTab函数做到这一点：

```
selectTab( tabsNav.find('a:first') );
```

9.3　让标签页更进一步

这一章描述了创建一个简单标签页组件涉及的步骤。但是，根据你的实现需要，还有其他一些方法可以用来修改或扩展标签页。这些方法包括书签和历史追踪支持，创建自动轮换效果，引用外部标签内容，以及将标签页作为一个手风琴组件（accordion widget）显示。

9.3.1　书签和历史（后退按钮）追踪

当你在增强体验里用标签条来控制大块的屏幕区域时，用户可能希望它扮演一个导航组件的角色，使浏览器的后退按钮遍历每一次标签点击，让它们返回之前的标签。这种记录和恢复某个特定标签视图的能力通常称为*深度链接*（deep linking），它是一种补充性功能，经常被包括在历史和后退按钮支持中。

1. 支持书签和深度链接

要在标签页组件里实现深度链接，需要在网页载入时检查URL#号串，如图9-6所示。如果它对应某个标签的ID，脚本就会自动选中并显示那个标签。

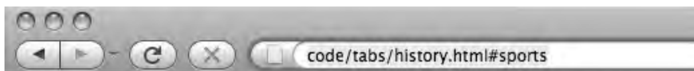


图9-6 URL里的#号串

URL#号串可以用window.location.hash这个JavaScript属性捕捉到一个变量里，我们会用它来寻找href属性匹配这个#号串值的标签。如果存在这样一个标签，就可以把它传递给selectTab函数。如果没有找到匹配的标签（或是因为URL里没有#号串，或是因为有#号串但不匹配任何一个标签），那么选中列表里的第一个标签作为备用措施：

```
//寻找一个href匹配URL#号串的标签
var hashedTab = tabsNav.find('a[href='+ window.location.hash +']');

//如果标签存在，选中它
if( hashedTab.size() > 0){
 selectTab(hashedTab);
}

//如果不存在，选中第一个标签
else {
 selectTab( tabsNav.find('a:first') );
}
```

现在，我们的标签组件具备在指定#号串时载入正确标签的能力了。当然，这种能力只有在标签被选中更新#号串时才有用。为此，我们会在selectTab函数的后部加上一行代码，用标签锚元素的href（对应其标签面板的id）来更新URL#号串：

```
 function selectTab(tab){
      tabsNav
          .find('li.tabs-selected')
              .removeClass('tabs-selected')
              .find('a')
                  .attr('tabindex','-1');
       tab
           .attr('tabindex','0')
           .parent()
               .addClass('tabs-selected');

       tabsBody
           .find('div.tabs-panel-selected'')
               .attr('aria-hidden',true)
               .removeClass('tabs-panel-selected');

            $( tab.attr('href') )
               .addClass('tabs-panel-selected')
               .attr('aria-hidden',false);

       // 更新#号串以匹配选中的标签
     window.location.hash = tab.attr('href').replace('#','');

     tab[0].focus();
    };
```

9

这行代码的添加位置刚好在给标签指派焦点之前，这样就能确保#号串更新后，焦点回到标签上（这对给键盘事件提供上下文环境来说是必需的）。

现在，这些标签会将它们的状态报告给URL#号串，让收藏和通过邮件发送标签成为可能。但是，在实现这一功能的过程中，我们引入了一些新的问题：首先，只要用户点击了某个标签，浏览器现在都会滚动到标签面板那里；其次，我们破坏了后退按钮的功能，它不能把用户导航到前一个选中的标签上！

要修复滚动问题，建议用JavaScript修改各个标签面板的id属性，使它们不再对应标签锚的href值。这样的话，URL#号串发生改变时，页面上不再有任何元素的ID匹配这个#号串，浏览器就不会滚动了。当然，修改这些ID时，标签页脚本也需要修改，改成点击某个标签时寻找这些新的ID。具体的做法是对selectTab函数做一个简单的调整，将它设置成寻找某个标签面板，其id等于#号串值加上你选择的任意预定义前缀。

但是，修复后退按钮就需要做一些额外的工作了。

2. 支持后退按钮

现在，每当用户选中一个标签时，URL会发生变化，浏览器则会把变化保存在它的历史菜单里。尽管如此，后退按钮却不会把用户带回他们最近访问过的那一页，甚至前一个标签也不行。事实上，它似乎什么事都不干，因为浏览器不会自动刷新到与最近的#号串关联的状态。

创建操作URL#号串的组件时，建议加入逻辑代码，只要#号串发生变化就更新组件，无论变化来自浏览器的后退按钮、前进按钮还是历史菜单都是如此。在下面这些情况下，追踪URL#号串状态是很有意义的：

- 当标签页包含一大块网页内容时（因为用户可能会认为这些标签页实际上是新网页）；
- 当用户经常会保存（或收藏）指向某个特定标签状态的链接时。

为那些在#号串发生变化时更新的组件提供支持应该是很直观的，但很不幸，事实并非如此，原因是浏览器支持各异。写作本书时，只有Internet Explorer 8和Firefox 3.6支持HTML5的hashchange事件，这种干净可靠的方法能探测URL#号串何时发生变化，然而其他所有的浏览器都不能原生支持这个事件，因此没办法知道#号串何时发生了变化。

为了支持其他浏览器，我们可以模拟原生的hashchange事件，方法是设置一个JavaScript函数定期检查URL#号串，并将它与之前检查时的值进行比较。如果值不同，#号串就有了变化，此时脚本可以调用一个函数，基于新的#号串来更新标签页。

```
//获得当前#号串
var hash = window.location.hash;

//检查#号串变动并相应地设置标签页的函数
function checkHash(){
  if(hash != window.location.hash){

     //#号串变了，相应地设置标签
     alert('THE HASH CHANGED!');

     //重置#号串变量
     hash = window.location.hash;
```

```
    }
}
//每1/2秒运行一次checkHash函数
setInterval(checkHash, 500);
```

在Internet Explorer 7和之前的版本中，用户按下后退按钮时，浏览器总是会向JavaScript报告它在网页载入时得到的原始URL#号串值，哪怕事实上它的地址栏里已经有了可见的变化，因此这些流行浏览器的#号串值是不可靠的。为了在这些版本的IE里支持后退按钮，开发者通常会在网页里插入一个`iframe`，用它的`src`属性追踪#号串变动，因为在点击后退/前进按钮时`iframe`的#号串能够可靠地发生变化。

这种实现后退按钮支持的方法是非常底层的，它的实际脚本编程不在本章的讨论范围内，但是一些现有的JavaScript库能服务于这个目的。其中一个这样的库叫做jQuery History（位于www.mikage.to/jquery/jquery_history.html）。这个插件用起来相当简单，它用一套方便的程序包同时支持后退按钮和书签，还处理了各种各样的浏览器怪癖和bug。本书所附的脚本将此插件作为一个可选项。9.4节将介绍如何使用它。

9.3.2 自动轮换的标签页

自动轮换标签页是另一种网页上常见的功能，它的每个面板都会显示一段时间，然后被另一个标签面板取代。这种流行功能出现在了新闻网站的首页上，用来循环显示一组当日热门文章。

这一章里的标签页几乎不用做多少扩展工作就能实现自动循环。脚本只需简单地选定初始标签页，然后设置一个间隔循环，在大约5秒钟后（视标签页内容的长度而定）切换到下一张标签页即可。

使用jQuery的话这个循环就会像下面这样：

```
//生成一个时间间隔，用于自动循环标签页
var tabRotator = setInterval(function(){
 //获得当前选中的标签
 var currentTabLI = tabsNav.find('li.tabs-selected');

 //获得选中标签之后的同级标签
 var nextTab = currentTabLI.next();

 //如果同级标签存在，选中它
 if(nextTab.length){
     selectTab(nextTab.find('a'));
 }
 //如果不存在，我们就处于标签条的末尾，选择第一个标签
 else{
     selectTab( tabsNav.find('a:first') );
 }
//每5秒钟（5000 ms）重新运行一次
}, 5000);
```

如果没有办法停止标签页循环的话，这个功能可能会打扰用户。停止自动循环的方法很简单，只需用`clearInterval(tabRotator)`语句清除之前设置的时间间隔即可。我们建议每当用户在标签

页控件里的任何位置点击、聚焦或按键时立刻运行这段代码，就像下面这个例子：

```
tabs.bind('click keydown focus',function(){
  clearInterval(tabRotator);
});
```

这一章所附的插件将此功能作为一个可选项。要了解细节，请务必阅读9.4节。

9.3.3　引用外部标签内容

我们的新闻标签页组件范例可以进行扩展，用Ajax抓取外部内容放进面板里。这个功能很实用，特别是在你把网站的顶部导航转换成一个标签页组件时，这样用户无须刷新整张网页就能阅读每个版块的内容。（这是个很好的例子，显示了支持历史追踪的意义所在）。

Ajax填充式标签页组件的标记与本章之前讨论的基础标记很相似，但有一点除外：无序列表里的链接会引用外部网页，而不是本地内容的锚（警告：我们通常建议至少让活动标签的内容随网页和那些链接一起输出，让基本体验里的用户能访问到它）。点击某个标签后，脚本根据链接的href属性生成一个Ajax请求，将外部内容抓取到某个标签面板里。

强烈推荐jQuery UI的标签页组件，它是已支持这种行为的jQuery插件里一个绝佳的代表。除了Ajax支持之外，jQuery UI的标签页还具有可折叠标签面板的能力，以及可以用ThemeRoller工具添加样式。可以在jQueryUI.com/download上下载这个标签页插件。

9.3.4　将标签页显示为手风琴组件

标签页和手风琴控件之间有非常多的共同之处，虽然可能因为视觉布局的差异而不太明显，但是它们的行为几乎完全相同的。两者都能切换内容面板的显示，而且每次只能查看一个面板。

从技术上看，这两种控件背后的标记也可以非常相似，但是源代码顺序根据实现方式的不同可能会有变化。举个例子，这一章展示的基础标记包括了一张无序列表，后面跟着所有的内容面板，而手风琴组件的源代码顺序可能会是内容头和内容块交替摆放，反映出它们的视觉呈现顺序。抛开源代码顺序不谈，手风琴组件的HTML属性可以和本章描述的标签页完全一致，包括 ARIA角色和状态。事实上，因为手风琴组件和标签页如此相似，所以ARIA甚至都没有为它设计一个角色。W3C推荐的手风琴组件实现方法实际上是使用标签页的角色。

一个遵循本章原则的优秀手风琴控件范例可以在这里找到：http://test.cita.uiuc.edu/aria/tabpanel/tabpanel2.php。

9.4　使用标签页脚本

本书所附的标签页演示和代码（网址是www.filamentgroup.com/dwpe）包含一个可重复使用且用法简单的插件：jQuery.tabs.js。它集成了本章概述的那些脚本，可在你的项目里使用。

要在网页里使用这个脚本，只需下载并引用标签页组件演示页里列出的那些文件，然后在标签内容的父容器上简单地调用tabs方法即可。举个例子，如果你使用的是本章提供的基础标记，

就可以在ID为featured的div上简单地调用tabs方法：

```
$('#featured).tabs();
```

标签内容的标记需要遵循与本章范例相同的结构：一个父标签容器存放锚链接组成的无序列表，后面跟着一个div，内含一组对应的标签内容面板。

标签页插件还有两个配置选项，用来支持可收藏的标签URL和后退按钮。要使用这些功能，需要加入对jQuery History插件脚本的引用。请参见历史标签范例页，了解如何正确引用所有必需的脚本文件。

在我们的标签页插件里，状态追踪功能默认是禁用的，不过可以通过trackState选项（它接受一个true/false值）启用它们：

```
$('#featured).tabs({
  trackState: true
});
```

启用状态追踪后，还需要提供服务器上某个空白网页的URL，用它追踪历史（请阅读jQuery History插件的文档，了解此处的更多信息）。srcPath选项的默认值是jQuery.history.blank.html，可以用任意空白文件路径覆盖它：

```
$('#tabs').tabs({
  trackState: true,
  srcPath: 'blank.html'
});
```


我们的标签页插件还支持自动循环，它默认是禁用的，不过可以通过autoRotate选项来激活，方法是设置一个你想要的时间间隔，以毫秒为单位：

```
$('#featured).tabs({
  autoRotate: 5000
});
```

现在你应该理解了如何创建一个功能完备且可访问的标签页组件。你能看到如何组织内容，以及添加关键的ARIA属性和键盘导航行为，使它真正对所有人可用。标签页组件可以用于或小或大的内容块，因此考虑用户对书签支持和后退按钮行为的期望，能大大改善用户体验。本章讨论了追踪井号串状态和保留后退按钮支持，它们的核心技巧和原则也适用于各种各样的交互型及Ajax驱动型组件。

第 10 章

工具提示

工具提示（tooltip）用于在界面里“按需”呈现内容（通常是次要的内容，例如图标按钮的简单文字描述，图表里某个数据点的细节或者表单里帮助性的字段操作说明），让用户在不打断阅读或操作的情况下获取更多的上下文信息。工具提示的内容一般会显示为一小块叠加层，当用户把光标放到某个元素上时出现。

大多数浏览器默认包含一个标准的工具提示功能：任何指派给HTML标签内`title`属性的内容都会渲染成一条工具提示，当用户将光标悬停在某元素上1秒钟左右显示。但是，这些标准浏览器工具提示具有固定的外观（文字大小通常是固定的，位于光标右侧一个黄色方块内），无法用CSS修改。它们的内容也有限制，必须是文本字符串，不能使用HTML标签、图像、背景图像或其他可以用来引入样式、结构或层级的格式。它们甚至都不支持换行！在一些浏览器里，工具提示不支持换行，所有文化都在一行显示。如果文化过长，可能会超出可见的浏览器窗口。

在许多情况下，一种更加健壮的工具提示是很有用的，其中包括下面这些情况。

- 拥有特色风格设计的网站，它包括了圆角、阴影、边框和背景等高级CSS格式，工具提示应当反映出现有的设计。
- 显示产品图像、富格式文本或者复杂布局的电子商务网站。举个例子，Netflix的电影提示包含一张照片，一段剧情简介和一张表格，表格里列出了导演、演员、格式、类型和用户评分。
- 服务于老年/少年用户或视力障碍用户的网站。在那里，显示字体更大、对比度更高的工具提示要比原生工具提示更有帮助。

本章会探索创建自定义工具提示的一些方法。我们会使用能实现各种自定义外观和行为的渐进增强技巧，并加入可访问性所需的语义标记结构。具体来说，我们会讨论如何装饰和显示一个基于`title`属性值的简单工具提示，之后则是一个更为复杂的工具提示，它的层级结构是基于网页内容或用Ajax从外部网页抓取来的。

10.1 X 光透视

就工具提示的目标设计而言，我们会考虑一个经常用到工具提示的案例：一张注册表单。为了保持表单外观的简单清爽，我们会使用工具提示选择性地提供注册人实现零错误注册所

需的全部信息，在他们将光标悬停在标签和链接上时显示。我们的目标设计如图10-1所示。

Register Now

All fields are required

Email Address

Password

We'll ask this question to confirm your identity if you forget your password.

Secret Question

Secret Answer

Register now

Privacy policy

Benefits of registering

图10-1　增强体验里的表单字段工具提示

这张表单里有两种不同类型的内容适合使用工具提示。

- 简短的字段操作说明，比如格式指南，或者关于每个文本框里数据使用方式的信息。
- “隐私政策”（Privacy Policy）和“注册好处”（Benefits of Registering）的内容。用户可能选择先查看它们，然后再进入下一步（请见表单底部的链接）。

在增强体验里，每个文本框的操作说明会放到工具提示里，在用户把光标悬停到该字段的标签上时显示。这一设计还包括一个位于各个标签右侧的小“i”图标作为视觉指示，表明有额外的信息可用。如果用户把光标悬停在这个图标上，工具提示也会出现。屏幕阅读器用户把焦点移到关联字段上时，工具提示还会被朗读出来。操作说明是纯文本的，不需要任何特殊的功能或行为。简单地说，它们的表现就像那些在指定`title`属性时出现的标准工具提示一样。因此，在基础标记里，我们会给每个`label`元素简单地添加一个`title`属性，然后为增强体验编写一小段脚本，用来解析各个`title`属性里的内容，并将其展现在一个自定义样式的工具提示里（字体和设计都更漂亮）。

位于表单底部的“隐私政策”和“注册好处”这两个链接，会分别显示一小段隐私政策声明和一大块解释注册好处的内容。我们会将这两种内容块都作为工具提示显示，因为它们是补充性的信息，不是填写或提交这张表单所必需的。

隐私政策的内容是一个介绍性的段落，之后跟着一张简单的项目符号列表，如图10-2所示。

图10-2　增强体验里的隐私政策工具提示

因为隐私信息既简短又重要，需要在注册时传达给大多数受众，所以在基本体验里，我们会把它放在基础标记页面里的表单正下方，如图10-3所示。

图10-3　基本体验里，隐私政策的内容显示在表单下方

基本体验里的“隐私政策”链接会是一个锚链接，用户点击时将页面向下滚动到隐私内容上。为此，我们会让此链接的href属性匹配隐私内容块的ID，再加上一个井号字符（href="#privacy"）。在增强体验里，我们会用一个脚本找到锚链接的href属性和它关联的内容，然后在用户将光标悬停到链接上时，把该内容作为一条工具提示显示出来。

最后同样重要的是，“注册好处”的内容由相当长的一组功能和好处介绍组成，还带有一张显示“免费加入”（Free to Join）的图片，如图10-4所示。



图10-4　增强体验里的功能和好处介绍工具提示

虽然对许多人有帮助，而且可能让人感兴趣，但它就完成表单而言并不像隐私信息那样关键。因此，我们在基本体验里会把好处的内容单独放在一个网页里（benefits.html），用户可以点击“注册好处”链接来访问它。又因为我们不想把用户带入死胡同，所以会确保那张单独的好处网页能链接回注册表单，如图10-5所示。

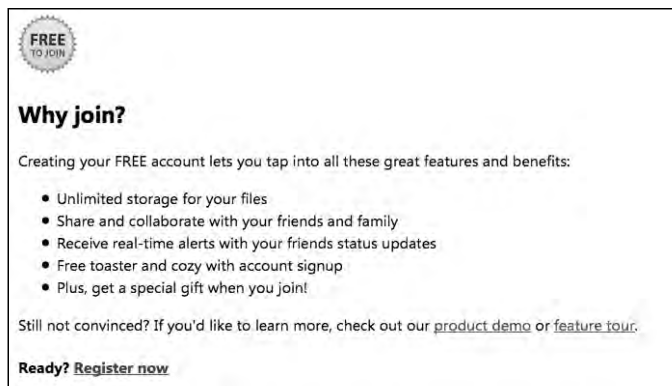


图10-5　单独一个网页里的功能和好处内容

在增强体验里，我们会用一个脚本找到链接的href值（benefits.html），然后生成一个Ajax请求来抓取该网页的内容，放入注册表单并显示为一条工具提示。

让人高兴的是，我们可以在这三种类型的工具提示里重复使用很大一部分的标记、样式和脚本。首先来看基于title的简单工具提示，然后概述用锚链接创建工具提示需要的修改，最后讨论如何用外部来源的内容创建工具提示。

10.2　用 title 内容创建工具提示

我们会先从最简单的工具提示开始：表单各个标签边上显示的字段操作说明文字。

10.2.1　基础标记和样式

注册表单里的每个字段都由一对label和input组成，label的for属性指向输入框的id以正确关联两者。input还带有一个text类，可以用它来应用样式规则：

```
<label for="email">Email Address</label>
<input type="text" name="user" id="email" class="text" />
```

我们会给每个label都添加一个title属性，内含操作说明，解释如何正确填写这个字段，或者提供有关数据使用方式的反馈：

```
<label for="email" title="To keep spammers out, we'll send a confirmation
email to make sure this is a valid email address">Email Address</label>

<input type="text" name="user" id="email" class="text" />
```

在大多数浏览器里，当鼠标移到标签上方时，title属性会显示为一条简单的工具提示，如图10-6所示。



图10-6　原生的浏览器工具提示在光标悬停时显示在标签下方

在基本体验里，我们可以给基础标记添加许多安全样式。为了让这张表单更容易阅读（以及更有吸引力），我们会设置全局字体样式：

```
body { font-family: "Segoe UI", Arial, sans-serif; }
```

接下来，我们会提供一些视觉线索，表明此标签有一条工具提示。我们会将样式属性cursor设置为help，使光标在悬停于标签上时看上去像个问号（？）。我们还会给标签文字加上点状的下划线，这是一种相当常见的Web惯例，用来表明存在工具提示，如图10-7所示。

```
label { cursor: help; border-bottom: 1px dashed #777; }
```

为了让表单更易于浏览，可以把标签堆叠到输入框上方，方法是将它设置为display: block，并添加一点顶部外边距：

```
input.text { display: block; margin: .5em 0 0; }
```



图10-7 应用安全样式后的基础标记

10.2.2 增强标记和样式

在增强体验里，我们会为表单和工具提示共同创建样式，全部三种类型的工具提示都可以借用它。

为了显示表单里有可用的工具提示，我们会在label右侧添加一个小“i”图标背景图像（代替在基本体验里使用的简单下划线）。我们会移除各个标签的底部边框样式属性，为图标指定一张背景图像，并添加右侧内边距来确保文字（即Email Address）不会遮住这个图标：

```
label {
  font-size:1.5em;
  text-align:left;
  margin-top:.8em;
  margin-bottom: .3em;
  border-bottom: 0;
  display: block;
  float: left;
  background: url(../images/icon-info.png) right 3px no-repeat;
  font-size:1.5em;
  padding-right:20px;
  text-align:left;
}
```

现在，表单字段看起来就像是我们的目标设计了，如图10-8所示。

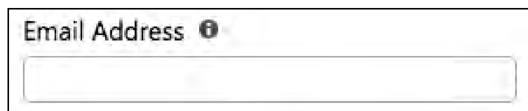


图10-8 应用增强样式后的邮箱表单字段

为了在增强体验里实现自定义的工具提示内容，需要生成一小段HTML并加上样式，使它看起来像目标工具提示设计。这段标记是一个带有tooltip类的简单div：

```
<div class="tooltip">To keep spammers out, we'll send a confirmation email
to make sure this is a valid email address</div>
```

这个工具提示的div会被加上一些样式，比如背景图像、边框和内边距，透明度则设置成90%，从而创建出一种半透明的效果（这个属性不能被旧版的Internet Explorer识别，但不会造成危害）。我们会把工具提示绝对定位到网页上，这样以后就可以用脚本动态设置顶部和左侧的坐标了：

```
.tooltip {
  position:absolute;
  background:#252122 url(../images/bg-tooltip.gif) top repeat-x;
  padding:12px 18px;
  font-size:1.2em;
  line-height:1.4;
  border:2px solid #fff;
  width:250px;
  color:#fff;
  opacity:.9;
}
```

为了进一步在视觉上打磨这个工具提示（而不需要使用多张背景图像），我们会添加一些浏览器专用和基于标准的CSS3属性：border-radius用于圆化边角，box-shadow用于在工具提示背后生成阴影。这些属性会显示在新版的IE、火狐、Safari和Opera浏览器上，如图10-9所示。在其他浏览器上，工具提示会显示出其他所有的样式规则，但边角是方形的，也没有阴影：

```
.tooltip {
  position:absolute;
  background:#252122 url(../images/bg-tooltip.gif) top repeat-x;
  padding:12px 18px;
  font-size:1.2em;
  line-height:140%;
  border:2px solid #fff;
  width:250px;
  color:#fff;
  opacity:.9;
  -moz-border-radius: 5px;
  -webkit-border-radius: 5px;
  border-radius: 5px;
  -o-box-shadow: 0 0 5px #aaa;
  -moz-box-shadow: 0 0 5px #aaa;
  -webkit-box-shadow: 0 0 5px #aaa;
  box-shadow: 0 0 5px #aaa;
}
```

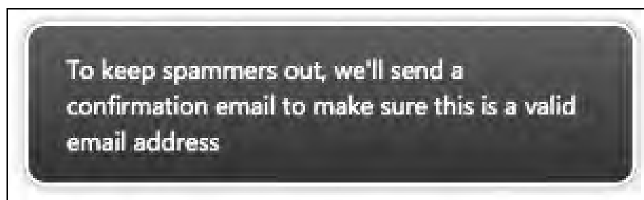


图10-9　应用样式后的工具提示增强标记

为了显示和隐藏工具提示，我们还会添加一个名为tooltip-hidden的类，用它把工具提示的

display属性设置为none，然后用脚本有条件地切换开启和关闭。

```
.tooltip-hidden { display: none; }
```

10.2.3 工具提示增强脚本

完成基础和增强标记以及样式之后，现在编写用于创建和显示工具提示的增强脚本。和样式一样，许多行为（包括容器定义和用来正确定位它们的鼠标事件）都可以一次编写，重复用于全部三种类型的工具提示。

首先，创建一些变量，用来生成增强的工具提示：关联邮箱文本input的label元素，标签title的值，以及一个指派给工具提示div的唯一ID，用来为屏幕阅读器用户关联标签和自定义工具提示：

```
var label = $('label[for=email]'); var tooltipContent = label.attr('title'); var tooltipID =
"email-tooltip";
```

现在，我们可以生成工具提示的标记了：一个新div元素，id为email-tooltip，内容来自标签的title（存储在变量tooltipContent里）。我们会给它指派tooltip和tooltip-hidden这两个类，这样就能用CSS修改这个div的样式，并确保它默认是隐藏的。我们还会给它添加tooltip这个ARIA role，向屏幕阅读器用户描述它的用途，另一个aria-hidden属性则告诉屏幕阅读器它当前是隐藏的：

```
var tooltip = $('<div class="tooltip tooltip-hidden" role="tooltip" id="'+ tooltipID +'"
aria-hidden="true">'+ tooltipContent +'</div>');
```

然后，我们会把工具提示附加到网页body的尾部：

```
tooltip.appendTo('body');
```

为了让屏幕阅读器识别这些工具提示，我们会给body元素添加application这个ARIA role：

```
$('body').attr('role','application');
```

我们已经生成了工具提示的标记，现在应该把label的title属性移除，以避免那些被复制的原生工具提示显示出来。为了让与label有关的新自定义工具提示能够被屏幕阅读器访问，我们会关联它们，方法是给标签指派aria-describedby这个ARIA属性，将它设置成与工具提示的id相同的值：

```
label.removeAttr('title').attr('aria-describedby', tooltipID);
```

接下来，我们会让工具提示在mouseover时显示，在mouseout时隐藏。我们会给label绑定一个mouseover事件，在悬停时显示工具提示，方法是移除它的tooltip-hidden类，将它的aria-hidden属性设置成false，并将工具提示的坐标定位到网页里光标位置的旁边。我们还可以通过向mouseover事件的回调函数传递一个变量参数（e）来获取mouseover事件的相关信息：比方说找出鼠标的x和y坐标，就像在下面代码里操作的参数e.pageY和e.pageX：

```
label.mouseout(function(e){
  tooltip.removeClass('tooltip-hidden')
      .attr('aria-hidden',false)
```

```
            .css({
                top: e.pageY - tooltip.outerHeight(),
                left: e.pageX + 20
            });
    })
```

我们会绑定一个mouseout事件，这样当用户将光标从标签上移开时，tooltip-hidden会被重新应用，aria-hidden属性则设置回true：

```
label.mouseout(function(){
  tooltip.addClass('tooltip-hidden')
    .attr('aria-hidden',true);
});
```

为了让工具提示对屏幕阅读器用户完全可访问，还必须做最后一步：确保与label关联的文本input具备aria-describedby属性。应用这个属性后，屏幕阅读器用户就可以访问到额外的描述性信息。根据屏幕阅读器的不同，工具提示会在屏幕阅读器用户将焦点放到input上时读出，或者会在用户按下组合键时读出。

```
$('#' + $(this).attr('for')).attr('aria-describedby', tooltipID);
```

现在，我们就有了一个基于原生title属性、功能完备的增强自定义工具提示，如图10-10所示。

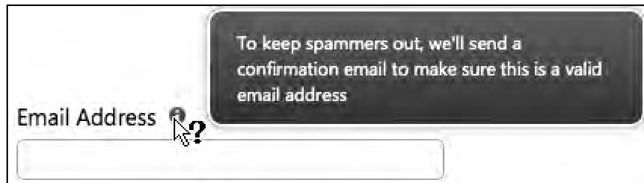


图10-10　增强体验里的工具提示

这种基于title的方法非常适合那些基于文本的简单工具提示，但无法用于文字格式更丰富的内容。接下来讨论的两种方法会演示如何用结构性内容创建增强提示，这些内容或者来自于网页上的不同版块，或者来自于某个外部资源。

10.3　用锚链接创建工具提示

我们经常使用本地锚链接提供从页面某一区域到另一区域的快速导航。为此，只需要把锚元素的href值设为井号加上网页目标元素的id即可。用这种方式设置href后，点击锚点会将焦点转移到对应的元素上，浏览器则会立刻滚动至那个获得焦点的元素（只要网页高度允许）。可以使用JavaScript和链接的href值找到网页上的关联内容，用它在增强体验里生成一条工具提示。

首先，我们将“隐私政策”内容写在基础网页标记里的注册表单下方，并指派一个值为privacy的id：

```
<div id="privacy">

  <h2>Our privacy policy</h2>
```

```
  <p>我们致力于保护你的隐私。这是我们对你的承诺。</p>
  <ul>
     <li>你的所有个人信息都会安全传输，并且以128比特加密存储</li>
     <li>我们绝不会将你的个人信息卖给第三方</li>
     <li>我们绝不会通过电子邮件向你发送营销信息或其他讨厌的垃圾邮件</li>
   </ul>
</div>
```

与它关联的链接在基本体验里是一个标准的本地锚点，将用户滚动到网页的这个版块上。为此，将“隐私政策”链接的href设置成引用隐私内容：

```
<a href="#privacy">Privacy policy</a>
```

在基本体验里，这个链接会将浏览器窗口滚动到隐私政策内容块上，如图10-11所示。

图10-11　基本体验里位于表单下方的隐私政策内容

把内容放入网页，关联锚和对应的内容块之后，就可以修改之前基于title的工具提示范例里的JavaScript，用链接的href值找到网页上的关联内容，然后生成一条工具提示。只需要修改脚本，更换我们将内容插入工具提示的方式。

首先，用tooltipID变量创建一个新的空白工具提示div：

```
var tooltip = $('<div class="tooltip" role="tooltip" id="'+ tooltipID +'"></div>');
```

因为任何本地锚点的href（井号加上id属性的值）和jQuery需要用来查找网页上工具提示内容的选择器，有着相同的语法（在此案例中是#privacy），所以脚本可以简单地引用整个href值，找到工具提示的内容：content:$(this).attr('href')。

剩下的就是把隐私内容附加到空白的工具提示里：

```
tooltip.append( $(this).attr('href') );
```

> **注意**　用这种方式附加内容时（无论是这里显示的jQuery append方法，还是标准的JavaScriptappendChild方法），它实际上会被从标记里原来的位置移到工具提示容器里，因此无须进行标记“清理”。

最后，借助现有的样式和脚本逻辑，在用户将光标悬停在“隐私政策”链接上时显示新的自定义工具提示，如图10-12所示。

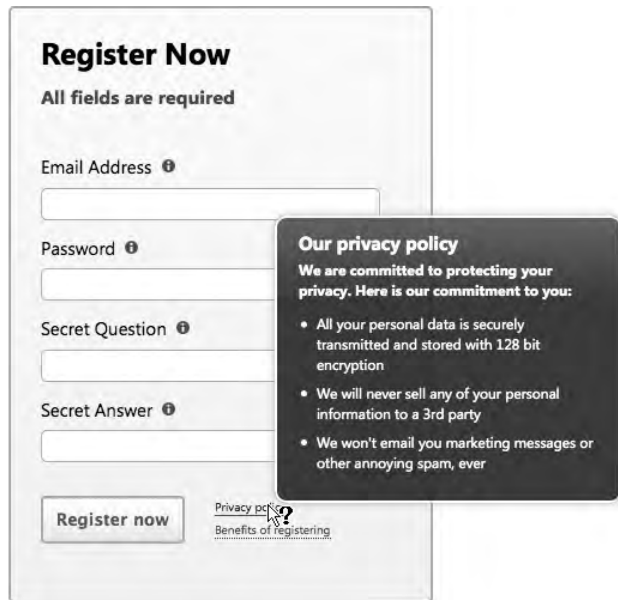


图10-12　增强体验里的隐私政策工具提示

10.4　用外部来源创建工具提示

我们已经展示了如何从某个title属性获取工具提示内容，以及如何利用本地锚链接的href值。现在来分析如何从一张单独的链接页里获取内容。

```
<a href="benefits.html">Benefits of registering</a>
```

在增强体验里，我们会用Ajax从服务器请求这张网页的内容，然后生成工具提示。理想情况下，Web服务器足够智能，能够判断是发送整张HTML网页，还是（当请求来自Ajax时）只发送生成工具提示所需的轻量级标记片段。如果我们有一台智能服务器被配置成当请求来自Ajax时用不同的方式处理，那么就可以利用jQuery的load方法，借助Ajax来请求那张好处页面，服务器则会明白只需返回网页内容的子集即可。load方法提供了一种极其简单的界面来用Ajax载入外部内容，只需要一个URL参数就能完成载入工作。

我们的增强脚本可以从链接的href里抓取URL，将它附加到工具提示的div上：

```
tooltip.load( $(this).attr('href'));
```

有些情况下这种智能服务器方式可能用不了，原因或者是你没有服务器的完全访问权限，或者是缺乏编写脚本逻辑的专业技能。在这些情况下，可以用Ajax抓取一整张href里引用的好处页（包括完整的文档head和body），然后编写JavaScript逻辑解析出文档的相关部分，将其插入到工具提示的div里。load方法提供了一种精妙的方式来抓取我们需要的标记，你只需要指出完整网页里我们需要的工具提示内容（即它的选择器：#content），之后jQuery就会将相关代码片段插入到我们的工具提示里，如图10-13所示。

```
tooltip.load( $(this).attr('href') +' #content');
```



图10-13 增强体验里的功能和好处工具提示

虽然这种只需要JavaScript的方式能够顺利工作，但是请求整张网页会带来显著的带宽和CPU开销，而我们真正想要的只是它的一小部分。在像这样的Ajax案例中，只要有可能，智能服务器方式永远应该是追求速度和效率的首选。

10.5 使用工具提示脚本

我们创建了一个jQuery插件：jQuery.tooltip.js。它能自动化创建工具提示的过程，而且能用于多种内容来源（title、锚点或外部网页）。这个脚本和范例代码可以在www.filamentgroup.com/dwpe上下载。

要使用这个工具提示脚本，只需用某个jQuery选择器找到网页里的一个或多个元素，然后对它们调用tooltip方法。举个例子，要给ID为#content的容器里某个段落中的所有链接赋予工具提示行为，可以使用下面这种引用方式：

```
$('#content p a').tooltip();
```

借助这一方法，脚本就会自动生成工具提示，然后用适当的内容填充它。

这个脚本足够智能，会寻找本章描述的三种可能内容来源（title属性、本地href锚点和Ajax外部资源），然后根据下面的逻辑生成工具提示。

(1) 它首先会看元素是否有一个自定义的HTML5 data-hrefpreview属性，如果有，就用Ajax从该属性引用的外部内容来源里抓取内容，并放入工具提示。data-hrefpreview属性是这个插件特有，在该脚本的上下文环境之外没有任何语义价值。

(2) 如果找不到data-hrefpreview属性，脚本便会寻找一个本地href引用（href="#privacy"），然后用此锚链接所引用ID里的内容生成工具提示。

(3) 如果这些来源都找不到，脚本会使用title属性来生成工具提示。

这套逻辑的顺序被设计成级联（cascade）的形式：data-hrefpreview属性的优先级高于href引用，后者的优先级又高于title属性。举个例子，如果某个链接包括了全部三种内容来源，脚本会使用data-hrefpreview属性引用的内容来生成工具提示：

```
<a href="#privacy" data-hrefpreview="privacy.txt" title="We respect your privacy but won't tell you specifics">Privacy policy</a>
```

这个脚本能实现优雅降级：如果某个元素调用了tooltip，但没有任何一种来源里有现成的内容，那么脚本就不给这个元素显示工具提示，却不会抛出错误。

通过运用渐进增强方法，就能利用原生的title属性、锚链接和HTML5的data属性，将网页语义标记里存储的内容转变成格式和样式丰富的工具提示。这种方法给了我们实现有效用户体验所需的设计灵活性，却无须牺牲基本用户体验的质量，也不会给可用性带来负面影响。

第 11 章

树形控件

树形控件（tree control）是在Web界面里表现层级信息的最流行方式之一：它们提供了一种方法，能在紧凑的空间里展示深度嵌套的多级内容，并允许用户有选择地展开和折叠节点，从而控制要显示树形结构的哪些部分。树形控件经常用于下列情况。

- 具有深度嵌套结构的网站导航组件，比如那些产品范围很广、层级很深的电子商务网站。
- 带有可展开/折叠行的数据表格，比如项目管理工具“微软项目管理”（Microsoft Project）里的表格。
- 带有分层文件夹的Web应用程序里的导航，比如电子邮件系统或者RSS阅读器。

树形控件本质上是一系列可折叠的嵌套容器，里面的父（或分支）节点能够展开，显示出一列子节点。树形控件的设计经常会采用一些方式来提示某些节点存在子节点，比如文件夹、加号（+）图标，或者容器节点标签左侧的箭头图标。

树形控件的关键特征是它内置类似应用程序的交互标准。具体来说，树形组件遵循传统的桌面应用程序行为，比如用Tab键将焦点从一个组件移到下一个上，以及用方向键管理单个组件内部的移动。（这种行为有别于标准的Web交互方式，后者用Tab键在各个可获得焦点的元素上顺序导航，将方向键留给屏幕滚动使用。）

随着越来越复杂的应用程序移植到网上，我们能看到Web行为和传统桌面行为之间的界线变得模糊起来。高级用户和屏幕阅读器用户习惯用键盘在树形控件里导航和展开/折叠节点，因此任何面向增强体验的自定义树形控件都应该包含标准的树形控件键盘导航惯例。

本章会展示如何在基本体验里构建一个使用语义化HTML的树形控件，然后再将它转变成一个能被屏幕阅读器访问和用键盘命令控制的交互式树形控件。

注意 许多网站使用的多级导航结构遵循传统的Web行为，也就是说用户可以按下Tab键将焦点从一个链接转移到另一个上。要创建这种类型的分级导航，可以考虑第8章描述的方式，用一系列可折叠的内容组件和关联它们的链接列表实现。

11.1 X光透视

我们的树形控件目标设计是一个基于Web的文档编辑工具里的导航面板，内含用户的文件

列表，如图11-1所示。

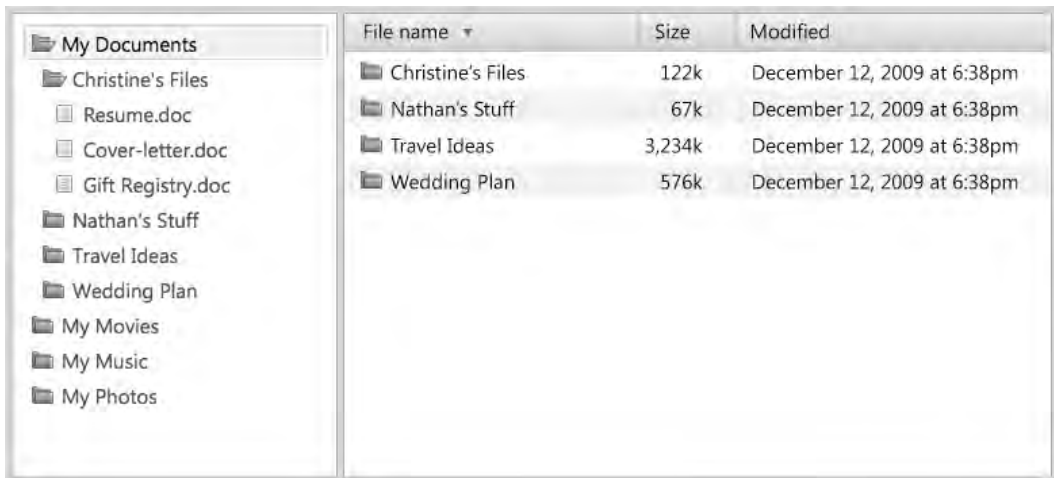


图11-1　文档编辑界面里的树形控件目标设计，左侧是导航树，右侧是预览面板

树形组件是这个双面板界面里的一部分，控制它右侧关联细节面板里的数据显示。鉴于X光分析和本章其余部分的需要，我们会重点关注树形组件。

树形组件里的节点分为两类：一种是组织内容的容器，用文件夹图标表示；另一种是只读或可编辑的文件，用页面图标表示。为了提供视觉反馈，每一个文件夹图标默认都是关闭的，在节点展开时则会变成打开的文件夹。

在增强体验里，用户可以通过鼠标点击或键盘命令来选择树形控件里的节点，这很像某个桌面搜索窗口。

- 用鼠标点击树结构里的某个文件夹会展开或折叠它，并在右侧面板里显示文件夹的内容预览。点击一个文件则会打开它以供查看和编辑。
- 使用上下方向键可以遍历整个树形结构，并用各个节点的信息更新预览面板。举个例子，用方向键将焦点移到某个文件夹节点上，会将预览面板更新为该文件夹的内容。
- 使用方向键聚焦某个文件夹或文件后，按下空格键或回车键会展开或折叠该文件夹，或者打开文件。按下右方向键或左方向键也能分别展开或折叠某个文件夹。

要让键盘命令能正确工作，必须为增强体验编写脚本逻辑，让树形组件里的当前选中节点获得焦点，这样它就能表现得更像一个桌面应用程序。用户可以（使用Tab键）让焦点跳到树形组件上，然后用方向键使焦点在各个树节点中转移。

在缺少JavaScript和高级CSS的基本体验里，树形控件表现为一张简单的链接列表，如图11-2所示。

基本体验里的用户同样可以通过鼠标点击和键盘控制来浏览整个树形控件，不过做法稍有不同：由于没有JavaScript，基本树形控件遵循标准的浏览器行为。他们可以点击文件或文件夹的链

接来打开它，或者用Tab（或Shift+Tab）键来遍历列表，然后按下回车键或空格键打开链接。用户打开某个文件夹链接时，页面会刷新显示该文件夹内容的快照以及树形控件的一小部分，以便快速导航回到完整列表，如图11-3所示。



图11-2　基本体验里的树形控件

从X光透视法看，文件和文件夹链接两者的标记选择都很明显，那就是锚链接（a），其中每个链接都指向一个具体的文件或文件夹快照页。要创建这种多级的组织结构，使用一张无序列表是一个良好的起点，因为它可以包含子无序列表来表示层级里的附加节点。



图11-3　在基本体验里查看树形控件的一部分

因此，我们会将树形控件的基础标记编写为一张由链接组成的无序列表。在基本体验里，应用尽可能少的样式，并显示链接的完整列表，不带有任何展开/折叠行为。这个页面对键盘用户和屏幕阅读器用户是完全可用和可访问的，因为他们只需像对待标准链接列表那样与树形控件进行交互。

在增强体验里，我们则会给列表项添加样式，用恰当的图标使它们看上去像个树节点，并且用JavaScript默认隐藏除顶级节点之外的所有节点。脚本还会应用逻辑来操纵浏览器焦点，这样用方向键就可以遍历整个树形控件，点击文件夹节点则会展开和折叠它们，如图11-4所示。

我们会指派ARIA属性来帮助增强体验里的屏幕阅读器用户理解这是一个树形控件，它的行为就像是一个桌面搜索工具，而不仅仅是一张无序列表里的一堆链接。增强脚本还会在用户与树形控件交互时正确指派和管理ARIA属性。

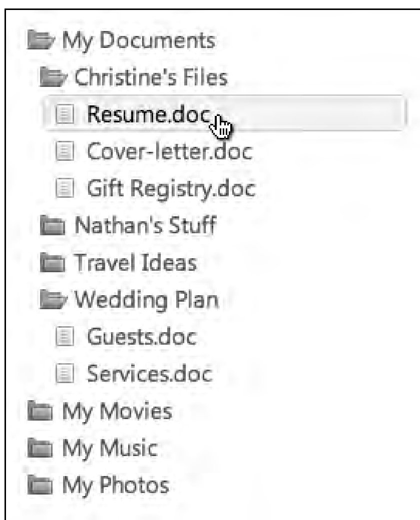


图11-4　增强体验里的导航树形控件

11.2　创建树形控件

首先，我们会为树形控件编写基础标记，通过使用一组嵌套的无序列表来创建恰当的文件夹与文件层级。然后应用样式增强，将这张列表转变成带有反馈图标的树形控件，并通过JavaScript应用浏览树形控件所需的鼠标和键盘行为。

11.2.1　基础标记和样式

我们的基础标记由嵌套的无序列表（ul）组成，它们包含的链接指向服务器里存放的文件和文件夹。

最外侧的列表带有files这个id，我们会使用它包含的4个顶级节点（列表项）作为文件夹来组织文件。在每个列表项中，文件夹名称是有链接的，这能让基本体验里的用户看到一张只列出某个文件夹所含文件的网页：

```
<ul id="files">
  <li><a href="documents/">My Documents</a></li>
  <li><a href="movies/">My Movies</a></li>
  <li><a href="music/">My Music</a></li>
  <li><a href="photos/">My Photos</a></li>
</ul>
```

在“我的文档”（My Documents）文件夹节点里的链接后面，我们会插入另一张无序列表，用于二级文件夹：

```
<ul id="files">
  <li><a href="documents/">My Documents</a>
```

```
        <ul>
        <li><a href="documents/Christines_Files/">Christine's Files</a></li>
        <li><a href="documents/Nathans_Stuff/">Nathan's Stuff</a></li>
        <li><a href="documents/Travel_Ideas/">Travel Ideas</a></li>
        <li><a href="documents/Wedding_Plan/">Wedding Plan</a></li>
        </ul>
    </li>
    <li><a href="movies/">My Movies</a></li>
    <li><a href="music/">My Music</a></li>
    <li><a href="photos/">My Photos</a></li>
  </ul>
```

用同样的方法填充剩余的树形控件层级，即把无序列表嵌套进文件夹节点，并将各个节点链接到其文件夹或文件路径上。

最后，在基本样式表里应用全局字体：

```
body { font-family: "Segoe UI", Arial, sans-serif; }
```

对于其他的样式而言，我们会让各种设备根据它默认的浏览器样式表来渲染这些链接和嵌套的列表。

在目前这个阶段，我们已经有了一个完美可用的基本体验导航系统，它利用了默认的嵌套列表缩进样式，并且以一种清晰的组织方式显示内容层级，如图11-5所示。



图11-5　基本体验里的完整导航树形控件

11.2.2　增强标记和样式

我们会添加若干属性到标记里，为每类节点（文件夹或文件）和它的状态（打开或关闭，鼠标悬停或带有焦点）应用视觉样式，并且指派ARIA属性来帮助向屏幕阅读器传达树形控件的角色和状态，以及它的各个组件。

首先，给body元素添加一个值为application的ARIA role属性，让屏幕阅读器识别出增强体验里的是应用程序控件，并让我们能够利用自定义键盘行为：

```
<body role="application">
```

我们会给最外侧的ul应用一个tree类。为了保持标记的简洁，默认将每个节点链接的样式设置为文件，随后脚本会识别出哪些节点链接了文件夹（它们包含子列表），并给它们的链接添加tree-parent类来分配文件夹图标：

```
<ul id="files" class="tree">
  <li>
     <a class="tree-parent" href="documents/">My Documents</a>
  <ul>
  <li><a href="documents/Christines_Files/" >Christine's Files</a>
      <ul>
      <li><a class="tree-parent" href="documents/Christines_Files/Resume.doc">Resume.doc</a>
...
```

我们想让屏幕阅读器将这一堆列表和链接当做一个树形组件，而非一张普通的链接列表，因此添加能够将它标识为树形组件的ARIA属性。（要查看树形组件角色和状态的完整参考文档，请访问www.w3.org/TR/wai-aria/roles#tree。）我们会给最外侧的无序列表添加tree这个role，给每个嵌套列表的role是group，每个列表项的role则是treeitem：

```
 <ul id="files" class="tree" role="tree">
   <li role="treeitem">
      <a class="tree-parent" href="documents">My Documents</a>
      <ul role="group">
         <li role="treeitem"><a href="documents/Christines_Files/"> Christine's Files</a>
             <ul role="group">
                 <li role="treeitem"><a class="tree-parent" href="documents
                 /Christines_Files/Resume.doc">Resume.doc</a></li>
...
```

aria-expanded属性标明某个节点处于展开还是折叠状态。默认情况下，每个节点都是折叠的，所以指派此属性时会用false值。我们还会（通过类）给折叠状态添加样式。网页载入时，每一个文件夹链接（tree-parent）都会带有tree-parent-collapsed类，而各个嵌套列表的类则是tree-group-collapsed。我们会切换这些类的开启与关闭，以此提供视觉反馈，并隐藏和显示各个节点。

一个折叠（隐藏）节点的标记看上去如下所示：

```
<li role="treeitem" aria-expanded="false">
  <a class="tree-parent-collapsed" href="documents/" tabindex="-1">My Documents</a>
```

```
  <ul role="group" class="tree-group-collapsed">......</ul>
</li>
```

节点展开时，增强标记会更新ARIA属性和类的值：

```
<li role="treeitem" aria-expanded="true">
  <a class="tree-parent" href="documents/" tabindex="0">My Documents</a>
  <ul role="group">...</ul>
</li>
```

我们还会给树形控件里的各个节点链接添加一个tabindex属性。默认情况下，一个网页里的所有链接都可以通过Tab键导航获得焦点，每次按下Tab键都会将焦点转移到下一个链接（或者其他可获得焦点的元素）上。但是，在这个案例里，我们想将树形控件看做单个组件，而不是一张链接列表，所以会给各个节点链接指派一个值为-1的tabindex，将其移出制表键顺序。然后，为了确保用户能用Tab键跳入跳出树形组件，我们会给树形组件里的某一个节点链接（默认是第一个）动态指派一个值为0的tabindex，使这个节点链接重新加入制表键顺序。随着用户用方向键浏览树形组件，之后获得焦点的节点链接将得到0这个tabindex值，而之前获得焦点的链接则恢复为-1，从而再次防止了它获得Tab键焦点。（本章后面分析增强脚本时会再次讨论这种称为*游动标签序号*的技巧。）

接下来，我们会给树形控件添加样式，比如设置字体大小，移除默认的列表项目符号，设置外边距和内边距，以及给树形控件里嵌套的各张列表设置8像素的左侧外边距，以缩进各个级别：

```
body {
 font-size: 62.5%;
}
.tree {
  overflow:auto;
  font-size:1.5em;
}
.tree,.tree ul,.tree li {
  list-style:none;
  margin:0;
  padding:0;
}
.tree ul {
  margin-left:8px;
}
```

树形控件里的所有节点链接都分配了文件图标的背景图像，调整样式后留出了足够大的左侧内边距，以确保链接文字不会遮盖背景图像。增强脚本会给文件夹节点特别指派一个tree-parent类，分配给这个类的是文件夹图标的背景图像：

```
.tree li a {
  color:#555;
  padding:.1em 7px .1em 27px;
  display:block;
  text-decoration:none;
  border:1px dashed #fff;
  background:url(../images/icon-file.gif) 5px 50% no-repeat;
```

```
}
.tree li a.tree-parent {
  background:url(../images/icon-folder-open.gif) 5px 50% no-repeat;
}
```

某个文件夹链接关闭时，它会分配到一个tree-parent-collapsed类，这个类会将背景图像换成关闭的文件夹图标，并把关联的子列表设为display:none来隐藏它，如图11-6所示。

```
.tree li a.tree-parent-collapsed {
  background:url(../images/icon-folder.gif) 5px 50% no-repeat;
}
.tree ul.tree-group-collapsed {
  display:none;
}
```

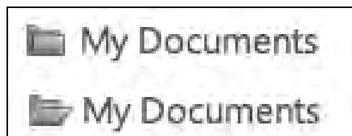


图11-6　折叠（顶部）和展开（底部）状态的文件夹节点

我们还想加入良好的视觉反馈，让鼠标和键盘用户了解焦点在哪个节点上。我们也会为鼠标用户特别提供一种悬停状态，标明他们的光标正处在哪个节点上。

用户让某个节点获得焦点的方式可以是点击它，也可以用Tab键进入树形控件，然后用方向键来遍历列表。点击或当前选中的节点会获得焦点，这样就为系统提供了所需的必要焦点，让它明白该如何执行后续的键盘命令，比如用回车键打开它。因此，带有焦点的样式相比悬停状态有着更深的轮廓和背景色。

当用户将他的光标悬停在某个节点上时，我们还提供了一种颜色更浅的悬停状态反馈，如图11-7所示。

```
.tree li a:hover,
.tree li a.tree-parent:hover,
.tree li a:focus,
.tree li a.tree-parent:focus,
.tree li a.tree-item-active {
  color:#000;
  border:1px solid#eee;
  background-color:#fafafa;
  -moz-border-radius:4px;
  -webkit-border-radius:4px;
  border-radius:4px;
}
.tree li a:focus,
.tree li a.tree-parent:focus,
.tree li a.tree-item-active {
  border:1px solid #e2f3fb;
  background-color:#f2fafd;
}
```

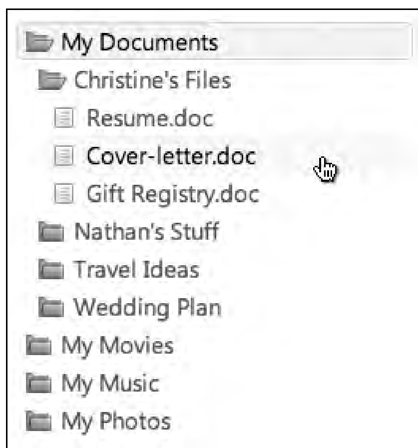


图11-7　“我的文档”文件夹节点获得了焦点，而Cover-letter.doc则显示了悬停状态

11.2.3　树形控件增强脚本

我们的增强JavaScript会给基础标记添加默认的ARIA role和class属性，创建展开、折叠和上下穿行树形控件的自定义事件，然后在用户通过鼠标或键盘命令与树形控件交互时触发这些事件。

1. 指派树形控件增强属性

脚本的第一步是引用即将增强成树形控件的无序列表，然后将它储存在一个tree变量里：

```
var tree = $('ul#files');
```

接下来，脚本会添加适当的ARIA role属性和状态类，并在标记里设置各个节点的tabindex：

```
//给树形控件添加角色和类
tree.attr({'role': 'tree'}).addClass('tree');

//将第一个节点的tabindex设置为0
tree.find('a:eq(0)').attr('tabindex','0');

//将其他所有tabindex设置设为-1
tree.find('a:gt(0)').attr('tabindex','-1');

//给所有ul添加group角色和tree-group-collapsed类
tree.find('ul').attr('role','group').addClass('tree-group-collapsed');

//为所有li子元素都添加treeitem角色
tree.find('li').attr('role','treeitem');

//找到树形控件里的“文件夹”项，折叠它们并添加tree-parent类
tree.find('li:has(ul)')
  .attr('aria-expanded', 'false')
  //找到li的直接下级锚链接
  .find('>a')
  .addClass('tree-parent tree-parent-collapsed')
```

现阶段，树形控件节点上的静态属性都已经设置完成了，接下来可以给脚本添加用户交互行为。

2. 应用树形控件行为

我们会创建自定义的jQuery事件来展开和折叠树形控件里的节点，并将它们绑定到树形控件本身（即最外侧的ul），从而利用JavaScript原生的事件指派行为。举个例子，当用户点击树形控件上的任一位置时，脚本会分析出点击的是哪个节点，然后根据它当前的状态展开或折叠它。这种绑定行为的方式（只对一个容器元素使用逻辑）比单独将事件绑定到各个树节点更有效率，执行速度更快。如果需要，还可以动态添加节点，因为每一个新节点都会自动继承指派给树形控件的行为。

首先创建expand事件。

```
//将自定义展开事件绑定到树UL
tree.bind('expand',function(event){
  //保存对点击目标链接的引用
  var target = $(event.target);

  //移除折叠的类，更换文件夹图标
  target.removeClass('tree-parent-collapsed');

  //获得目标同级元素的UL
  target.next()
      //用display:none隐藏UL并移除隐藏的类
      .hide().removeClass('tree-group-collapsed')

      //向下滑动展开UL
      .slidedown(150, function(){
          //找到上一级的LI，设置它的展开属性
          target.parent().attr('aria-expanded', 'true');

          //移除来自动画的内联样式
          $(this).removeAttr('style');
     });
});
```

现在，可以通过触发expand事件展开任意一个文件夹节点：

```
tree.find('a.tree-parent').trigger('expand');
```

接下来，创建一个collapse事件，它会逆向执行expand事件的操作：

```
//给tree UL绑定自定义的折叠事件
tree.bind('collapse',function(event){
 //保存对点击目标链接的引用
 var target = $(event.target);

 //添加折叠的类，更换文件夹图标
 target.addClass('tree-parent-collapsed');

 //获得目标同级元素的UL
 target.next()

 //向上滑动折叠UL
```

```
.slideUp(150, function(){
   //找到上一级的LI，设置它的展开属性
      target.parent() .attr('aria-expanded', 'false');

      //给UL添加隐藏的类，并移除内联样式
      $(this).addClass('tree-group-collapsed').removeAttr('style');
  });
});
```

这样，可以通过触发collapse事件来折叠某个打开的树节点：

```
tree.find('a.tree-parent').trigger('collapse');
```

有了这些可供支配的事件后，就可以让树形控件对鼠标点击做出反应，展开或者折叠。为了确保树形控件展开或折叠正确的节点，我们会检查被点击的元素是否是一个带有tree-parent类的链接，以及它的上级列表项是否有一个设置成true或false的aria-expanded属性：

```
//给tree绑定点击事件
tree.click(function(event){
  //保存对目标（即被点击的元素）的引用
  var target = $(event.target);

  //检查目标是否是一个树节点
  if( target.is('a.tree-parent') ){

      //检查树节点的上级LI是否是折叠的
      if( target.parent().is('[aria-expanded=false]') ){
          //对目标调用expand函数
          target.trigger('expand');
      }

      //如果不是，上一级必然已经展开了
      else{
          //折叠树节点
          target.trigger('collapse');
      }

      //将焦点移至目标节点
      target[0].focus();

     //防止浏览器追踪链接
     return false;
  }
});
```

注意 用户点击树形控件里某个不带有tree-parent类的链接时（换句话说，是一个文件节点），我们不会执行任何操作。相反，我们会让浏览器用原生方式处理这个点击，打开选中的文件。

当点击树形控件，或者用户用Tab键进入树形控件时，某个树节点就会获得焦点。如果树形控件里有节点获得焦点，那么就会用一些方式来管理聚焦状态。

- 用户在各个树节点里导航并移动焦点时，我们会使用游动标签序号技巧，将0 tabindex动态移动到获得焦点的项上。
- 用聚焦状态来确定用户按键时对应的节点，这样脚本会知道该展开或折叠哪个节点。
- 控制各个节点获得焦点的顺序，这样用户就可以用上下方向键来上下穿行整个树形控件结构。

首先，将聚焦节点的tabindex设置为0，这样就可以用Tab键访问它了（树形组件则通过它间接实现）：

```
//给tree绑定焦点事件
tree.focus(function(event){
  //如果存在之前获得焦点的树节点，将tabindex设为-1并移除活动的类
  tree.find('[tabindex=0]')
      .attr('tabindex','-1')
      .removeClass('tree-item-active');

 //指派0 tabindex给获得焦点的项，并添加活动的类
 $(event.target)
     .attr('tabindex','0');
     .addClass('tree-item-active');
});
```

注意 管理制表键焦点有一种替代方法，它借助名为aria-activedescendent的ARIA属性，为那些理解ARIA的最新款屏幕阅读器处理焦点。这种方法的缺点是它只适用于最新款的屏幕阅读器，而游动标签序号技巧适用于所有屏幕阅读器，对所有的键盘用户都适用。

我们会创建一个keypress事件（用左右方向键展开和折叠某个获得焦点的文件夹节点），然后将它绑定到树形控件的ul上。它关联的函数会根据事件的keyCode属性（37是左方向键，39是右方向键）检查按下的是哪个键。当按下左方向键或右方向键时，脚本会对获得焦点的项分别触发expand或collapse事件：

```
//给tree UL绑定keydown事件处理程序
tree.keydown(function(event){
  //保存对获得焦点项的引用
  var target = $(event.target);

  //如果是左方向键且列表是展开状态
  if(event.keyCode == 37
     && target.parent().is('[aria-expanded=true]')){
         //触发折叠事件
         target.trigger('collapse');

        //返回false以防浏览器滚动
        return false;
 }

 //如果是右方向键且列表是折叠状态
```

```
  if(event.keyCode == 39
      && target.parent().is('[aria-expanded=false]')){
          //触发展开事件
          target.trigger('expand');

          //返回false以防浏览器滚动
          return false;
  }
});
```

为了实现上下方向键导航，我们会再绑定两个自定义事件（traverseUp和traverseDown），它们会使焦点在树形控件里上下移动。

当用户在树形控件里上下导航时，我们想让紧挨着的可见节点获得焦点，无论它是否处于树形组件里的相同层级都是如此。举个例子，如果当前焦点是在“出行设想”（Travel Ideas）上，那么用户一次次按下上方向键时，在树形控件里位于它上方的每一行都会依次获得焦点，如图11-8所示。

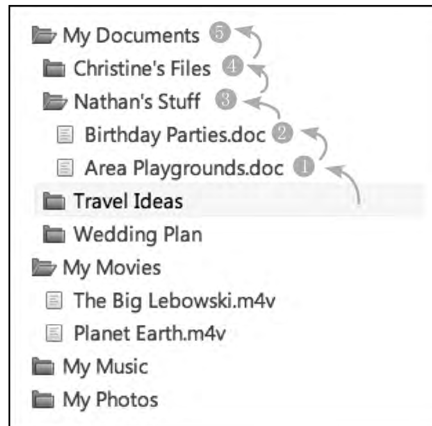


图11-8 在树结构里向上穿行的示意图

用于向上穿行的逻辑必须处理几种情况：用户应该能在同级节点中穿行，返回父节点，以及深入树形结构。

```
//给tree绑定自定义的traverseUp事件
tree.bind('traverseUp',function(event){

  //保存对事件目标（即树形控件里获得焦点的项）的引用
  var target = $(event.target);

  //保存对目标上级li的引用
  var targetLi = target.parent();

  //检查目标列表项之前是否存在同级项
  if( targetLi.prev().length ){
      //如果是，那么检查该同级项是否是展开的
      if( targetLi.prev().is('[aria-expanded=true]') ){
      //深入进去，将焦点移至它最后的可见子项上
```

```
            targetLi.prev()
                .find('li:visible:last a')[0].focus();
      }

          //如果不是，那么之前的同级项是折叠的
          else{
              //将焦点赋予之前的同级项自身
              targetLi.prev().find('a')[0].focus();
          }
      }
      //如果不是，那么不存在同级项，该项在自己的树群组里

      else {
          // 向上移动一级，尝试让上级节点获得焦点
          targetLi.parents('li:eq(0)').find('a')[0].focus();
      }
    });
```

有了上面这段代码，焦点位于树形控件内时，可以在用户按下上方向键后触发traverseUp事件：

```
//给tree UL绑定keydown事件
tree.keydown(function(event){
  //如果是上方向键
  if(event.keyCode == 38){
      //在获得焦点的项上触发traverseUp
      $(event.target).trigger('traverseUp');

      //返回false以防浏览器滚动
      return false;
  }
});
```

我们还会创建一个traverseDown事件，毫不奇怪的是，它基本上就是反过来的traverseUp。我们会检查获得焦点的元素是否是展开的，如果是，那么焦点应该跳到它的第一个子树项上。如果不是展开的，我们会检查它之后是否有同级项可以获得焦点。如果答案是false，那么当前获得焦点的项就是它树群组中的最后一个。我们会尝试寻找一个拥有后续同级项的上级列表项。如果能找到，就让它获得焦点。

如果这些条件都不为真，原因是用户已经穿行到了树形控件的末尾，那么很简单，traverseDown就不会把焦点移出当前获得了焦点的项。

```
//给tree UL绑定traverseDown事件
tree.bind('traverseDown',function(event){
  //保存对事件目标（即树形控件里获得焦点的项）的引用
  var target = $(event.target);

  //保存对目标上级li的引用
  var targetLi = target.parent();

  //检查目标上级的LI是否是展开的
  if(targetLi.is('[aria-expanded=true]')){
      //它是展开的。深入进去，将它的第一个子项获得焦点
      target.next().find('a')[0].focus();
  }
```

```
    //如果不是展开的，检查是否有后续的同级项
    else if(targetLi.next().length) {
        //让它后续的同级项获得焦点
        targetLi.next().find('a')[0].focus();
    }

    //如果不是展开的，且没有后续同级项，那它就是此树群组列表的最后一项
    else {
      //尝试找到拥有后续同级项的上级li，将焦点移到那里
      Targetli.parents('li').next().find('a')[0].focus();
    }
});
```

现在，可以在按下下方向键时触发traverseDown事件了：

```
//给tree UL绑定keydown事件
tree.keydown(function(event){
  //如果是下方向键
  if(event.keyCode == 40){
      //在获得焦点的项上触发traverseDown事件
      $(event.target).trigger('traverseDown');

      //返回false以防浏览器滚动
      return false;
  }
});
```

增强脚本完成后，这个树形控件就具备了完备的功能，兼容ARIA，并且可以通过键盘访问。

11.3 使用树形控件脚本

本书所附的演示和代码（位于www.filamentgroup.com/dwpe）包含一个脚本：jQuery.tree.js。它以一种简单易用的方式运用了本章概述的这些原则。

要在你的网页里使用这个脚本，只需下载并引用树形控件范例页里列出的那些文件，然后等DOM就绪后在网页的任意列表元素上调用tree方法。举个例子，要创建一个基于本章基础标记的树形控件，需要找到一个id为files的ul元素，然后对它调用tree方法：

```
$('#files').tree();
```

还可以同时在若干个多级列表上调用tree方法，把它们都转变成树形控件。举个例子，这段代码会把#content容器里的所有ul变成树形控件：

```
$('#content ul').tree();
```

这个插件的设计很简单，只包含了一个选项，用于将某个节点（或多个节点）默认设置成打开状态。只需把一个或多个列表项选择器传递给expanded选项即可。举个例子，如果想默认打开第一个节点：

```
$('#files').tree({
  expanded: 'li:first'
});
```

或者展开一个ID为wedding的列表项：

```
$('#files').tree({
  expanded: '#wedding'
});
```

还可以打开多个节点，方法是给若干列表项指派一个class，并在expanded选项里引用这个class：

```
$('#files').tree({
  expanded: '.default-expand'
});
```

树形控件的增强版组件可以用简单的HTML语义在基础标记里非常清晰地描述出来。这一方法的优点在于，增强体验可以在坚实的基础标记之上，通过简单叠加额外的ARIA属性、CSS样式和JavaScript行为实现。它是教科书式的渐进增强范例。

tabindex属性和键盘导航都是重要的功能，让所有用户都能高效使用树形控件，还能让屏幕阅读器访问。编写支持这些行为的脚本所需花费的额外时间必不可少，因为仅仅依靠鼠标输入无法服务于所有人，甚至还会让一些用户感到不爽，因为他们期望树形控件能像原生的操作系统控件那样工作。

第 12 章

HTML5 canvas图表

以图形化方式表现复杂数据（比如饼图、折线图、面积图或柱状图）是一种强有力的方法，它不仅能传达信息，还能阐明那些单凭仔细观察大型数字表格几乎不可能发现的模式与趋势。我们看到越来越多的图表和图形出现在网上，它们用于传递关键信息，包括：

- 统计数字和百分数的比较，例如投票或民意测验的结果；
- 跟踪一定时间内的简单活动或性能信息，例如访问网站的用户数量，关于某个主题的新闻报道次数或者股票价格；
- 在显示复杂的金融或科学趋势数据时直观表现移动平均数、最小值和最大值之间的相关性，以及其他可视化形式的增强分析。

呈现可视化图表和图形数据有两种最常见的方式，一种是嵌入由Web服务器生成的静态图表图像，另一种则需要Flash、Java或Silverlight等插件在浏览器里生成图表。可惜，这两种方式都不够理想：静态图像的问题在于它们只是简单的图片，不包含任何实际的数据值。（对那些无法访问可视性内容的用户来说，唯一能为他们总结“图片”中数据的方式是生成`title`属性，但这种方式很难掌控，哪怕数据集非常简单也是如此。）另外，虽然基于插件的图表解决方案可以实现少部分桌面浏览器的访问，但对许多移动设备、旧款浏览器或者出于安全原因限定浏览器设置的企业环境来说，它们可能不支持所需的特定插件版本，因此要么通知用户进行升级，要么在图表应该出现的位置显示一片空白。

2004年的夏天，Safari浏览器的开发小组提出了另一种方法，他们引入了一个新的HTML元素，名为`canvas`（画布）。`canvas`元素提供了一套原生的JavaScript绘图API，可以在前端生成视觉图像。2009年，HTML5规范将`canvas`纳入标准元素，而它早已在新版的火狐、Safari、Chrome、Opera浏览器以及Webkit驱动的移动设备（比如iPhone、Android和Palm Pre）上得到了广泛支持。再加上相对简单的Internet Explorer变通方法（这一章会讨论），`canvas`能安全用于所有的流行现代浏览器。

`canvas`元素的可视化能力不仅在图表和图形上得到了体现，它可以绘制任何类型的图像，甚至还支持动画。然而，需要重点注意的是，它自身并不能算做是一种完全可访问的解决方案。像`image`元素一样，它是一种纯视觉性的格式，对盲人等残障用户或者搜索引擎这样的非人工数据解析器来说，它几乎没有能传达自身内容的可访问式方法。

但是，`canvas`却有能力读取结构性数据（以HTML `table`的形式）并对其进行视觉性渲染，

从而在浏览器客户端上创建出可视化的图表。这种表现内容的新方法既具有丰富的视觉性，又可供所有用户访问。它提供了数据的增强化视觉再现，同时又保证了信息的传达，不会把任何用户拒之门外。

本章会演示如何用canvas元素根据HTML table生成一张折线图，并重点强调一些为有效使用canvas而需要理解的关键概念。随后，12.4节会相对简单地讨论一些更高级的逻辑，它们是创建其他某类图表（包括折线图、饼图、柱状图、堆叠式柱状图和面积图）所必需的。

注意　下面叙述的标记和脚本代表并阐明了用canvas元素正确实现可访问图表用到的关键概念和原则，但这里并不包括实现能用的可访问图表所需的完整框架。要使用一套可行的解决方案，建议你从www.filamentgroup.com/dwpe下载visualize.js插件，然后运用下面的指导原则，根据你的喜好对它进行定制。

12.1　X 光透视

假设我们有一套企业“仪表盘”，上面跟踪记录了员工的销售业绩数据。我们希望能将销售数据显示为一张折线图，这样用户就能看到每个人在各个产品类别方面的销售情况，发现趋势，还能很容易地比较业绩。我们希望这张数据图表看上去如图12-1所示。

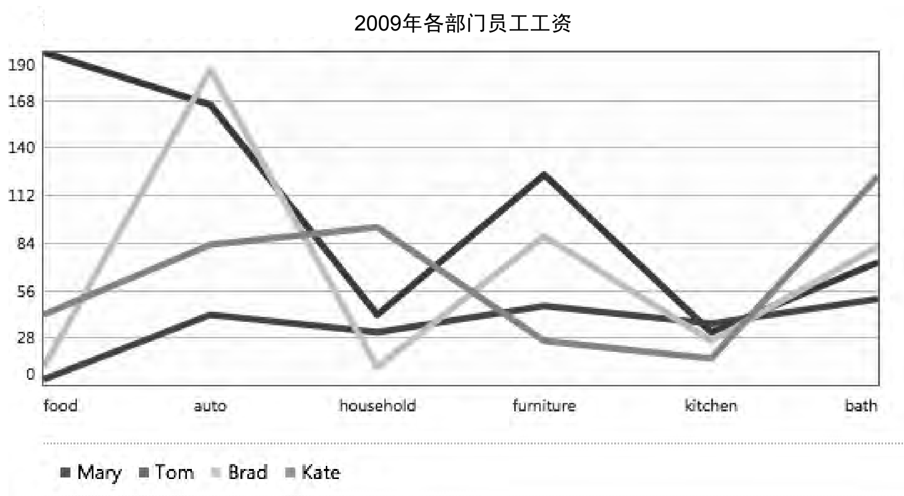


图12-1　销售业绩图表的目标设计

用X光透视这张图表时，我们发现图像中的颜色和线条粗细并不是用户需要看到的重要“干货”，数据才是。因此我们会从一张包含所有销售信息的简单HTML表格起步，而不是在网页里嵌入一张图表的图片，如图12-2所示。

2009年各部门员工工资

	food	auto	household	furniture	kitchen	bath
Mary	190	160	40	120	30	70
Tom	3	40	30	45	35	49
Brad	10	180	10	85	25	79
Kate	40	80	90	25	15	119

图12-2 表格形式的业绩数据

因为这张表格是用结构良好的HTML编写的，所以它的数据任何人都可以访问，包括移动设备、屏幕阅读器和搜索引擎。

相比本书里的其他组件，创建可访问的图表需要多花几步，但是它仍然遵循相同的基本模式：首先创建基础标记，以完全可访问的格式包括用户需要的所有相关信息；然后在增强体验里应用高级CSS和JavaScript，将基本数据转变成增强图像。

为了创建基于表格的图表，我们会解析HTML `table`，提取出所有数据值，并在浏览器里用`canvas`元素根据这些数据创建图表。我们会一一介绍用表格生成折线图的相关步骤，然后在本章末尾讨论如何使用jQuery Visualize插件。它是一个功能完备的图表制作脚本，能在这本书的网站上找到。

12.2 基础标记

在制作图表之前，需要一个数据来源，所以我们一开始会在基础标记里构建一张HTML `table`。

这张表格包含了4位员工（Mary、Tom、Brad和Kate）的零售数据和他们在各类百货商店（食品商店、汽车商店、家居商店、家具商店、厨房用品商店和浴室用品商店）中的近期销售量。

表格可以拥有许多子元素和属性，它们会给视觉性和非视觉性用户两者都增加语义价值。我们会利用所有能编码进基本版网页的结构和意义。完成标记后，销售数据的表格看起来如下所示：

```
<table>
  <caption>2009 Employee Sales by Department</caption>
  <thead>
      <tr>
          <td></td>
          <th scope="col">food</th>
          <th scope="col">auto</th>
          <th scope="col">household</th>
          <th scope="col">furniture</th>
          <th scope="col">kitchen</th>
          <th scope="col">bath</th>
      </tr>
  </thead>
  <tbody>
```

12

```
        <tr>
            <th scope="row">Mary</th>
            <td>190</td>
            <td>160</td>
            <td>40</td>
            <td>120</td>
            <td>30</td>
            <td>70</td>
        </tr>
        <tr>
            <th scope="row">Tom</th>
            <td>3</td>
            <td>40</td>
            <td>30</td>
            <td>45</td>
            <td>35</td>
            <td>49</td>
        </tr>
        <tr>
            <th scope="row">Brad</th>
            <td>10</td>
            <td>180</td>
            <td>10</td>
            <td>85</td>
            <td>25</td>
            <td>79</td>
        </tr>
        <tr>
            <th scope="row">Kate</th>
            <td>40</td>
            <td>80</td>
            <td>90</td>
            <td>25</td>
            <td>15</td>
            <td>119</td>
        </tr>
    </tbody>
</table>
```

这张表格的标记由3个主要部分组成：caption（它是表格特有的一种元素，提供了对后续表格数据的简短介绍）、thead（表格顶部水平方向的表格头行），以及tbody（包含表格内容主体的那些行）。我们用th元素标记每列和每行的第一项，通过应用值为col或row的scope属性，说明这些内容分别用做相应列或行的标题。scope属性能够被新款的屏幕阅读器识别（比如JAWS的最新版），但是，在范围更广的屏幕阅读器上，这一属性的支持程度是不一致的。尽管如此，我们仍然觉得把它放进标记里对任何阅读标记的人都有帮助，包括屏幕阅读器用户和开发者等。

注意　如果是更复杂的表格，比如表格头和单元格横跨多行或多列，请使用headers属性来关联表格头和它们的数据单元格。第3章介绍了如何使用这个属性。另外要注意，本章描述的脚本需要修改才能适用于如此复杂的表格结构。

虽然这张表格的语义很丰富,但是在大多数浏览器里,表格元素的默认外观还很不尽如人意,如图12-3所示。在Firefox 3等浏览器里，表格不带边框，单元格内边距也不大，这让阅读数据变得困难。

2009年各部门员工工资

	food	auto	household	furniture	kitchen	bath
Mary	190	160	40	120	30	70
Tom	3	40	30	45	35	49
Brad	10	180	10	85	25	79
Kate	40	80	90	25	15	119

图12-3　不带样式的基础表格标记

为了改善基本表格的可读性，我们会把一些“安全”样式添加到基本样式表里。绝大多数老款和移动浏览器都支持这些样式，而且会在不支持它们的浏览器里安全地降级。首先，我们会合并表格的边框，在单元格之间加一条线，并给单元格应用浅灰色的边框和一些内边距，让表格行更容易浏览。caption元素同样也是表格的一个视觉标题，所以我们会加粗文字并添加底部外边距使它变得突出。我们还会添加一堆font-family，让表格的字体与网站其他部分看上去一致，但不会定义字体大小，而是让各种设备使用它自己的默认值：

```
body {
  font-family: "Segoe UI", Arial, sans-serif;
}

table {
  border-collapse: collapse;
}

caption {
  margin: 0 0 .5em; font-weight: bold;
}

td, th {
  text-align: center;
  border: 1px solid #ddd;
  padding:2px 5px;
}
```

在给基本CSS添加这些细微样式调整之后，数据表格现在更易于阅读了，如图12-4所示。

2009年各部门员工工资

	food	auto	household	furniture	kitchen	bath
Mary	190	160	40	120	30	70
Tom	3	40	30	45	35	49
Brad	10	180	10	85	25	79
Kate	40	80	90	25	15	119

图12-4　添加“安全”样式后的基础表格标记

12.3　创建可访问的图表

用基础标记里的数据表格创建图表分为以下几步：解析基本标记里的数据以供可视化使用；初始化canvas元素并编写指令，绘制展现其内部数据值、图例（legend）和标签的图像；如果将数据表格留在页面上效果更好，就添加增强样式来改进它的显示方式；最后，添加一些指令和快捷方式来支持可访问性。

12.3.1　解析表格数据

建立和美化HTML表格之后，我们会用JavaScript采集它的信息并组织成可以用来生成图表的格式，此图表会按部门显示每个人的销售量对比情况。

首先，创建一个名为tableData的JavaScript对象，用来完整保存所有信息：

```
var tableData = {};
```

这个tableData变量可以作为附加变量的一个命名空间，扮演一个干净容器的角色，存放从表格解析出的数据。举个例子，可以声明tableData.myNewProperty给这个对象添加一个属性，然后任意设置这个属性的值。要生成折线图，先从HTML表格里收集必要的信息，将它存储在tableData对象里。

把表格元素存入一个名为table的变量以方便引用：

```
var table = $('table');
```

1. *X* 轴标签

tableData对象的第一个属性会命名为xLabels。顾名思义，它会包含那些沿着底部*X*轴的标签值，它们的作用是帮助识别图表上方的那些数据点。

就这张图表而言，我们会让“部门”（department）数据从左到右显示，使*X*轴标签映射到表格顶部thead里的那些内容，如图12-5所示。

2009年各部门员工工资

	food	auto	household	furniture	kitchen	bath
Mary	190	160	40	120	30	70
Tom	3	40	30	45	35	49
Brad	10	180	10	85	25	79
Kate	40	80	90	25	15	119

图12-5　*X* 轴标签从THEAD中的表头读取

我们会将xLabels定义为一个空数组，然后使用jQuery的each方法轮流访问thead里的各个th元素，把它们的文本值插入xLabels数组：

```
//定义xLabels数组
tableData.xLabels = [];
```

```
//轮流访问表格thead里的各个th元素
table.find('thead th').each(function(){
  //把每一个th里的文本值插入xLabels数组
  xLabels.push( $(this).html() );
});
```

2. 图例标签

表格左列（员工姓名）里的数据会被用作图表里的图例，如图12-6所示。

2009年各部门员工工资

	food	auto	household	furniture	kitchen	bath
Mary	190	160	40	120	30	70
Tom	3	40	30	45	35	49
Brad	10	180	10	85	25	79
Kate	40	80	90	25	15	119

图12-6 图表的图例标签从表格TBODY里的TH中获得

要捕捉这些值，我们会给tableData对象添加另一个名为legend的属性：

```
//定义legend数组
tableData.legend = [];

//轮流访问表格tbody里的各个th元素
table.find('tbody th').each(function(){
  //把每一个th里的文本值插入legend数组
  legend.push( $(this).html() );
});
```

可以看到，生成图例数组的过程与创建xLabels数组所用的循环方式非常相似，但它引用的是tbody里的th元素，而不是thead里的。

3. *Y*轴标签

接下来是图表的数字型*Y*轴标签，可以使用下面的程序逻辑计算出表格主体数据集里的最高值和最低值，效果如图12-7所示。

```
//定义初始值为0的topValue和bottomValue属性
tableData.topValue, tableData.bottomValue = 0;

//轮流访问tbody里的各个td元素
table.find('tbody td').each(function(){

  //定义thisValue变量，存放转换成数字的td文本
  var thisValue = parseFloat( $(this).text() );

  //检查thisValue是否比当前的topValue更高
  if( thisValue > tableData.topValue ) {
      //thisValue成为新的topValue
      tableData.topValue = thisValue;
```

```
    }
    //检查thisValue是否比最低值更低
    if( thisValue < tableData.bottomValue ){
        //thisValue成为新的bottomValue
        tableData.bottomValue = thisValue;
    }
});
```

2009年各部门员工工资

	food	auto	household	furniture	kitchen	bath
Mary	190	160	40	120	30	70
Tom	3	40	30	45	35	49
Brad	10	180	10	85	25	79
Kate	40	80	90	25	15	119

```
topValue = 190
bottomValue = 3
```

图12-7　通过确定最高和最低数据值计算出Y轴

现在已经给tableData对象定义了两个新值：topValue和bottomValue，分别用于最高和最低数据点。我们会用它们计算出Y轴的标签。在这个案例里，柱状图或折线图所用的Y轴需要覆盖从3（bottomValue）到190（topValue）的范围，因此脚本会将此轴设为0~190，并在图表高度能够容纳的范围加入适量的刻度，如图12-8所示。

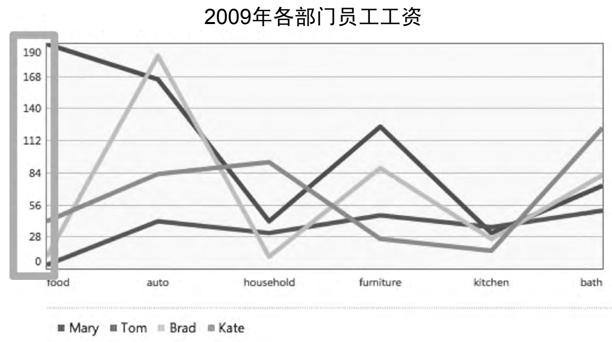


图12-8　折线图上显示的Y轴

为了集齐生成Y轴标签所需要的信息，我们需要确定几项数据：

- 图表的高度；
- Y轴上各个标签之间理想的像素距离；
- 数据集里最低值和最高值之间的范围。

我们会根据表格的宽度和高度设置图表的尺寸：

```
//用表格元素的高度定义chartHeight变量
var chartHeight = table.height();

//顺便也定义一下chartWidth，以备将来使用
var chartWidth = table.width();
```

然后，计算出显示在Y轴上的标签数量。我们会用chartHeight除以一个代表标签间像素距离的数。在这个案例里，我们会指定30像素作为理想的标签间距，因为考虑到字体大小，这个数字能够创建出充足的距离。把结果四舍五入后，就获得了显示在Y轴上的标签数量：

```
//根据chartHeight定义numLabels变量
var numLabels = Math.round(chartHeight / 30);
```

了解要显示的标签数量之后，我们会找出数据集的整体范围，然后在这个范围内迭代生成各个标签的值。定义变量totalRange作为最高和最低值之间的差值。虽然在我们的样本数据里totalRange与topValue相等，但是这一步计算仍然是必需的，因为它确保了那些包含负值的数据集能正确体现在整体范围里：

```
//定义totalRange作为最高和最低值之间的差距
var totalRange = tableData.topValue - tableData.bottomValue;
```

我们已经有了足够的数据来生成Y轴标签，现在定义一个新的yLabels数组并填入标签值。标签值遍布最低值到最高值的区间，数量则是我们能放进去的最大值（考虑到图表的高度）：

```
//定义yLabels数组
tableData.yLabels = [];

//找出递增的量，如果不是整数则增加到整数
var yIncrementAmt = Math.ceil(totalRange / numLabels);

//定义一个递增变量，从最低值起步
var currentValue = tableData.bottomValue;

// 如果currentValue仍然小于最高值减去递增量
while( currentValue < tableData.topValue - yIncrementAmt){
   //把currentValue添加到yLabels数组中
   tableData.yLabels.push(currentValue);

   //使currentValue增加一个递增量
   currentValue += yIncrementAmt;
}
//最后，添加最高值作为最后的标签
tableData.yLabels.push(tableData.topValue);
```

4. 数据组

接下来创建的属性是dataGroups，这个二维数组聚合表格里各个单元格（td）的值，然后根据我们选择的图表数据显示方向，把这些数据按行或列组织起来。这里会按照从左到右的顺序，把这些值根据包含它们的tr行进行分组，如图12-9所示。

需要解析表格数据才能实现图12-9显示的数组。为此，定义dataGroups数组，然后遍历表格里的每一行，并把每一行的文本值数组作为dataGroups里的一项：

```
//定义dataGroups数组
tableData.dataGroups = [];

//遍历tbody里的各个tr
table.find('tbody tr').each(function(i){
```

12

```
    //将dataGroups的下一项定义为一个新数组
    tableData.dataGroups[i] = [];

    //遍历这个tr里的各个td
   $(this).find('td').each(function(){
       //将td里的文本转换成数字
       var tdNumberValue = parseFloat( $(this).text() );

      //将值添加到dataGroups[i]数组
      tableData.dataGroups[i].push( tdNumberValue );
    });
  });
```

2009年各部门员工工资

	food	auto	household	furniture	kitchen	bath
Mary	190	160	40	120	30	70
Tom	3	40	30	45	35	49
Brad	10	180	10	85	25	79
Kate	40	80	90	25	15	119

```
tableData.dataGroups = [
[190,160,40,120,30,70],
[3,40,30,45,35,49],
[10,180,10,85,25,79],
[40,80,90,25,15,119] ];
```

图12-9 创建数据组的方法是解析每一行的数据值，并将其放入一个数组

我们现在有了一个由程序生成的数组来匹配表格里的内容行，接下来就是有趣的部分了：绘制图表。

12.3.2 用canvas实现数据可视化

我们有了必要的数据，可以开始绘制图表了。第一步是创建canvas元素，并用之前（基于表格尺寸）创建的chartHeight和chartWidth变量来设置它的width和height属性：

```
//创建一个新的canvas元素
var canvas = $('<canvas />');

//设置宽度和高度属性
canvas.attr({
 height: chartHeight,
 width: chartWidth
});
```

注意 我们用HTML的width和height属性（而不是CSS）设置canvas的大小，这一点很重要。虽然可以使用CSS给canvas元素添加样式，但是canvas的绘图API只关联该元素的width和height属性。

创建出一个空白的canvas之后，会把它添加到网页中一个包装器div里。这个div包含了和图表有关的许多元素，包括标题、说明和轴的标签。我们会创建这个包装器div，设置它的尺寸以匹配canvas，再把canvas插入它里面，然后把它添加到网页中，紧跟在表格后面：

```
//创建canvas容器div
var canvasContain = $('<div class="chart"></div>')
  //设置它的CSS宽度和高度属性
  .css({width: chartWidth, height: chartHeight})

  //给它添加canvas元素
  .append(canvas)
  //将它插入到表格元素后面
 .insertAfter(table);
```

1. 让IE跟上时代

我们差不多已经准备好绘制图表了。在此之前，需要说明一个情况：Internet Explorer（直至IE 8）不支持canvas元素和它的绘图API。为了让图表能在Internet Explorer上工作，我们需要一种变通方法。幸好，谷歌公司开发了一个名为exCanvas的库，它将canvas的命令翻译成VML，后者是一种只有Internet Explorer才支持的私有绘图语言。要使用exCanvas，需要在网页的head里引用excanvas.js脚本，而且必须放在其他任何使用canvas命令的脚本之前。可以用一个条件注释来引用这个脚本，以确保只有IE能看到和下载它：

```
<!--[if IE]><script src="excanvas.js"></script><![endif]-->
```

挂接上exCanvas后，我们只需要在IE里调用G_vmlCanvasManager.initElement方法就能初始化canvas。我们会把它放进if语句里，检查exCanvas是否已定义，这样它就只会应用到合适的浏览器（Internet Explorer）上：

```
//通过检测excanvas对象判断浏览器是否是IE
if( typeof(G_vmlCanvasManager) != 'undefined' ){
  //如果是，就初始化canvas，这样就能在IE里绘图了
  G_vmlCanvasManager.initElement( canvas[0] );
}
```

现在，就可以在canvas上绘制图表了。

2. 绘制图表线条

我们会使用canvas的2D绘图API来绘制图表，所以第一步是调用getContext方法，将画布的上下文环境定义为2D。我们会用ctx这个变量来保存对此上下文环境的引用，它用来给画布发送绘图指令：

```
//定义ctx变量作为canvas 2D上下文环境对象
var ctx = canvas[0].getContext('2d');
```

我们希望图表上的各线条之间有所区别，因此创建一个包含十六进制颜色值的数组。为简单起见，我们称它为colors：

```
var colors = ['#be1e2d', '#666699', '#92d5ea', '#ee8310'];
```

注意 在挑选用于图表的颜色时，请注意选择能被约占5%的色盲用户（大多数是男性）辨别的颜色。要避免选择具有相同明度（即颜色深度等级）的红色、绿色和黄色，并应该在色盲模拟器上测试你所选的颜色，比如Color Oracle（http://colororacle.cartography.ch）。

12

开始绘制线条：默认情况下，canvas绘图API的坐标方向是从左上角开始往右下角形成正值。但是，图表里的坐标是从左下角开始，然后向右侧和上方扩展，因此需要重置默认的起始点。我们会用canvas的translate方法将*Y*轴的值重置为我们的画布高度，用的仍然是chartHeight变量：

```
ctx.translate( 0, chartHeight );
```

现在，绘图的起始点是图表的左下角。

之前我们给tableData对象创建了一个dataGroups属性，它包含了来自表格的数据，按行进行分组。现在，使用一个for循环来遍历dataGroups数组并绘制图表线条。作为循环运行前的准备，我们会计算绘制画布上各个点时应该使用的坐标增量，用画布的宽度除以每条线上的点数，并把商保存在一个名为xIncrementAmt的变量里。我们还会用lineWidth属性定义图表线条的宽度为2像素：

```
//定义xIncrementAmt来表示折线图里的点距
var xIncrementAmt= chartWidth / (tableData.xLabels.length -1);

//设置画布上下文环境的lineWidth为2 px
ctx.lineWidth = 2;
```

定义完这些变量后，就准备好循环遍历这个数据数组并实际绘制线条了。首先移动绘图工具到当前群组里的第一个数据点，方法是使用ctx.moveTo(0,-points[i])，然后使用迭代变量i来匹配线条颜色和颜色数组中的项，该颜色数组与该群组对应。接下来，用beginPath方法把绘制工具放置在画布上，然后循环遍历群组里的每个点，从上一个点到当前点之间绘制一条线，再使用xIncrementAmt变量的数值增加左侧偏移量。

循环结束后，连接各点的线条就已经绘制完成了。我们会给线条应用颜色和宽度，然后关闭路径结束绘制（否则它会在后面的循环里连接到下一条线），如图12-10所示。

```
//在dataGroups数组上开始for...in循环
for(var i in dataGroups){
  //定义points为当前的dataGroups子数组
  var points = dataGroups[i];

  //把画笔移动到此数组的第一个数据点上
  //用负数的Y值从左下角向上移动
  ctx.moveTo(0,-points[i]);

  //设置线条颜色为颜色数组里的对应颜色
  ctx.strokeStyle = colors[i];

  //按下画笔开始绘图
  ctx.beginPath();

  //定义currentXvalue存放增加值
  var currentXvalue = 0;

  //开始用for循环遍历数据点数组以绘制各个点
  for(var j in points){
```

```
      //从当前位置画一条到下一个点的直线
      ctx.lineTo(currentXvalue,-points[j]);

      //给currentXvalue加上增加量
      currentXvalue += xIncrementAmt;
    }
    //给线条描边，使其可见
    ctx.stroke();
    //完成这条线段，抬起画笔
    ctx.closePath();
  }
```



图12-10　在Canvas图表里，每个数据集都以唯一的颜色绘制

3. 添加坐标轴线条和标签

困难的部分已完成：我们的图表画出来了！

接下来，给坐标轴添加文字标签，让用户能够理解图表线条的含义。之前我们为tableData对象创建了xLabels和yLabels两个属性，现在用它们在图表坐标轴沿线设置标签。

从技术角度看，可以用多种方式创建这些标签。也许最清爽的方式是用canvas原生的文本方法把文字标签直接写到画布上。但不遗憾，canvas文本方法并不像canvas绘图方法那样得到广泛的支持，所以canvas文本不符合我们的需求。

相比将标签写入画布，另一种可行的方法是添加所需文字的HTML标记，然后用CSS将其定位到图表div之内。我们会为每一组标签各自创建一张无序列表（ul），然后循环遍历各个属性来给标签添加列表项。为了给这些标签添加样式和定位，我们会混合使用外部和内联的CSS。和所有的CSS布局一样，我们会尽可能把CSS都放在外部样式表里，但是对这些用JavaScript动态定位的标签来说，更合理的方式是内联设置坐标，而不是试图去猜测它们可能的位置，然后应用各种类。我们会绝对定位这些标签，并根据坐标轴的不同，只内联设置各个列表项的left或bottom值。

我们会用JavaScript生成这张列表，循环遍历xLabels数组并用程序设置各个列表项的left和width内联样式，让它们沿*X*轴平均分布。生成的*X*轴标签HTML如下所示，效果如图12-11所示。

```
<ul class="labels-x">
  <li style="left: 0px; width: 83px;">food</li>
  <li style="left: 83px; width: 83px;">auto</li>
  ...
</ul>
```



图12-11　添加*X*轴标签后的图表

*Y*轴标签略微复杂一些，因为水平线需要在各个标签增量上横穿图表，方便联系线条和它们的数据点。我们会为*Y*轴标签使用下面的标记：

```
<ul class="labels-y">
  <li style="bottom: 199px;">
      <span class="line"></span>
      <span class="label">190</span>
  </li>
  <li style="bottom: 166px;">
      <span class="line"></span>
      <span class="label">145</span>
  </li>
  ...
</ul>
```

可以看到，*Y*轴标记里的每个列表项都包含了两个span：一个用于图表线，另一个用于文字标签。这一次，我们使用bottom而不是left来定位这些列表项，这样项目就会从下往上垂直堆叠。

剩下的就是给标签和线条添加样式，可以用下面这些样式进行处理：

```
.chart { position: relative; }
.labels-x,
.labels-y {
  position: absolute;
  left: 0;
  top: 0;
  list-style: none;
  margin: 0;
  padding: 0;
  width: 100%;
  height: 100%;
}
.labels-x li,
.labels-y li {
  position: absolute;
  bottom: 0;
  color: #555;
}
.labels-y li span.line {
  position: absolute;
```

```
  border: 0 solid #ccc;
}
.labels-x li {
  margin-top: 5px;
}
.labels-y li {
  width: 100%;
}
.labels-y li span.label {
  right: 100%;
  position: absolute;
  margin-right: 5px;
}
.labels-y li span.line {
  border-top-width: 1px;
  width: 100%;
}
```

我们把各个标签ul的尺寸设置为图表容器div自身width和height的100%，这样就能很容易地使用top、right、bottom和left属性将列表项定位在图表各处。然后，可以绝对定位这些列表项（以及标签，在*Y*轴上），根据坐标轴的不同给它们设置100%的top或right值。

我们还会把*Y*轴列表项（以及它们带有line类的子span）的宽度设为100%，这样就可以给这些span添加一个顶部边框来制作线条，对应*Y*轴上的各个标签，如图12-12所示。

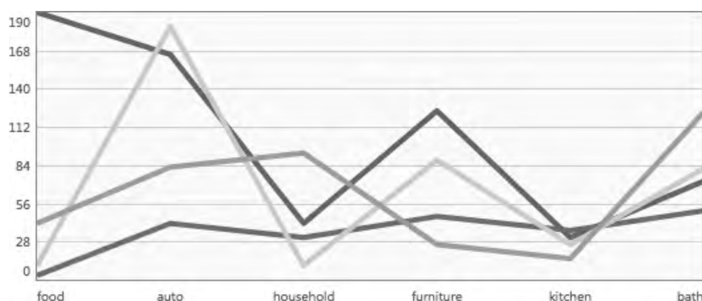


图12-12 带有*X*/*Y*轴标签和水平线的图表

4. 添加颜色图例

我们的折线图还差一点就能完成：需要一个图例来将每条线关联到一名员工身上。

这个图例也会是一张简单的无序列表，里面的每一项都包含一个span作为员工姓名边上的颜色图标。我们会遍历legend数组，用下面的脚本生成一张列表（注意，我们还引用了colors数组里序号相同的项，获得每位员工的对应颜色）：

```
//创建图例UL
var legendList = $('<ul class="legend"></ul>');

//遍历legend里的项
for(var i in tableData.legend){
  //用legend项里的文本创建列表项
```

```
  $('<li>'+ tableData.legend[i] +'</li>')
   //前面加上颜色值方块的span
   .prepend('<span style="background: '+
   ➥tableData.colors[i] +'" />')

   //把列表项附加到图例列表中
  .appendTo(legendList);
}
//把图例UL附加到图表容器div中
legendList.appendTo(canvasContain);
```

提示　上面的脚本被设计成遍历和生成整个无序列表后才开始向网页插入内容。虽然可以先创建无序列表并将它插入网页，然后再填充它，但是这么做可能会引发浏览器的性能问题，对非常大的表格来说尤为如此。首先生成完整的列表能确保脚本快速执行。

现在，用CSS给图例添加样式。我们会使它保持简单，将它以横向列表的格式放置在图表下方。只需用到下面的CSS代码：

```
ul.legend {
  list-style: none;
  position: absolute;
  border: 1px solid #000;
  padding: 10px;
  left: 100%;
  margin-left: 10px;
}
ul.legend li span {
  width: 6px;
  height: 6px;
  float: left;
  margin: 3px;
}
```

带着样式的图例现在位于图表下方，如图12-13所示。

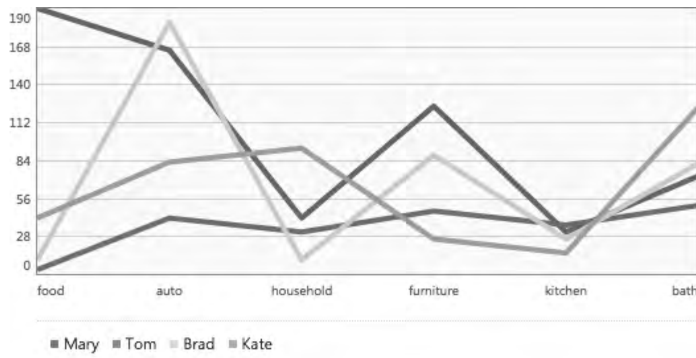


图12-13　图表里的图例用颜色编码的方块来标明各条线

5. 添加图表标题

可视化表格数据的最后一步是复制来自表格的caption元素并给它添加样式，用作图表的标题：

```
//创建图表标题div
$('<div class="chart-title">'+ table.find('caption').html() +'</div>')
  //把它附加到图表包装器div前部
  .prependTo(canvasContain);
```

然后给标题添加样式，使它和图表看起来协调：

```
.chart-title {
  font-size: 14px;
  font-weight: bold;
  position: absolute;
  bottom: 100%;
  margin-bottom: 10px;
  width: 100%;
}
```

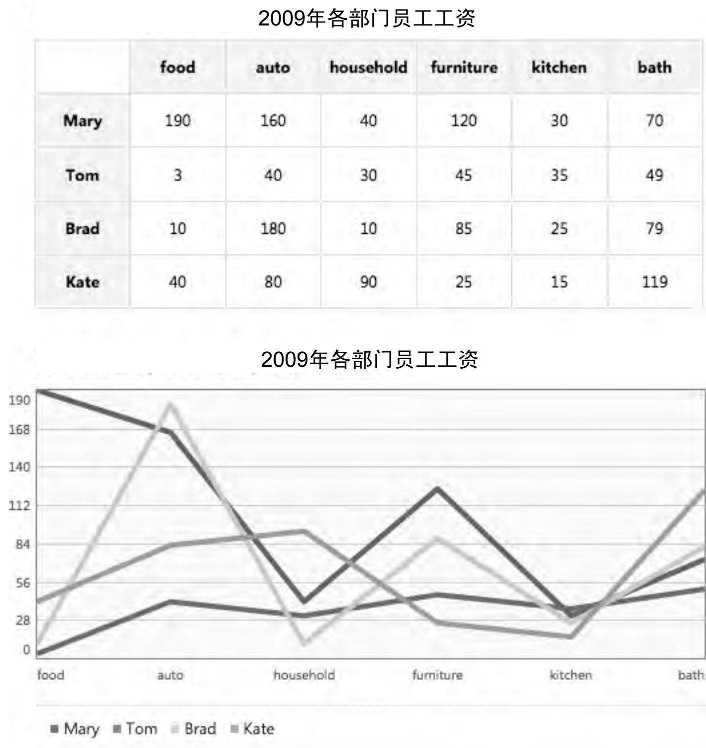
应用上所有样式后，我们的最终canvas图表如图12-14所示。

图12-14　完成后的图表包括一个基于表格caption的标题

12.3.3　添加表格增强样式

尽管我们有了一张漂亮的新图表，但在某些情况下，可能有必要保留网页上的原始表格，比如额外提供易于浏览的数字式数据。在这个案例里，我们会给增强样式表添加一些规则来改进它的显示方式。

我们会把为基础标记创建的样式作为基础，修改caption元素的样式使其变大变粗，并给表格头添加一层非常浅的灰色背景色，将它们与表格数据单元格区分开，如图12-15所示。



2009年各部门员工工资

	food	auto	household	furniture	kitchen	bath
Mary	190	160	40	120	30	70
Tom	3	40	30	45	35	49
Brad	10	180	10	85	25	79
Kate	40	80	90	25	15	119

图12-15　添加了样式的数据表格和最终的折线图，两者堆叠在一起，同时可见

12.3.4　保持数据的可访问性

本章前面提到，我们的网页是从一张可访问的数据表格起步。既然没有做过任何事危害到用户访问表格的能力，我们的数据就仍然是可访问的，只不过多了一张canvas图表作为补充。但是，鉴于网页上同时显示着表格和图表，我们想将两者之一隐藏起来，也许是为了给小屏幕上的布局释放一些空间，或者只是为了减少页面冗余。

1. 将表格隐藏起来

隐藏数据表格的常用方法包括使用CSS的`display: none;`属性，或是用JavaScript把它从网页里完全删除。很遗憾（也许还很明显），这两种技巧不仅将内容从视觉上隐藏起来，对屏幕阅读器用户同样也隐藏了，这样实际上就背离了采用表格生成方式的目的，也就是可访问性。

幸好，还有其他一些方法能在不影响可访问性的情况下隐藏内容。我们建议确定这张表格的绝对位置，把它远远地放到网页左边沿之外。这样做能成功使它对视力正常的用户隐藏，但它仍然存在于文档流中，能够被屏幕阅读器访问：

```
table { position: absolute; left: -99999px; }
```

2. 对屏幕阅读器隐藏图表

我们生成的这张图表完全是为了改善视觉外观，所以它对那些视力受损的用户价值很低。因此，对屏幕阅读器用户隐藏这一部分内容会有帮助。

W3C的WAI-ARIA规范草案提供了role="img"这个属性，它非常适合这一种用途，因为它告知屏幕阅读器这个元素扮演的是一张图像的角色。

就像图像有alt属性一样，img角色可以和aria-label属性搭配起来，用纯文本描述图像。我们会把这些属性添加到图表包装器div上，后者包括了canvas和其他列表：

```
<div class="chart" role="img" aria-label="这张折线图所表现的数据来自：2009年员工销售量表（按部门分类）">
  <!—canvas and the extra lists go here -->
</div>
```

这些属性用jQuery很容易添加：

```
//在我们的包装器div上设置role和aria-label属性
canvasContain.attr({
  'role': 'img',
  'aria-label': 'Chart representing data from: ' + table.find('caption').text()
});
```

许多屏幕阅读器用户仍然在使用不能很好地支持（或者完全不支持）ARIA的屏幕阅读器版本，所以我们会提供备份文字，提醒这些用户此元素包含的内容纯粹是为了视觉用途，并提供一个链接让用户跳过该元素。这段消息会放在一个角色为img的div容器里，这样使用新版屏幕阅读器的用户就会知道这个div纯粹是视觉性的，他们不会看到里面包含的消息。

首先，向图表div插入一个段落元素，它包含一段描述内容视觉性本质的消息：

```
$('<p class="chart-access-message"><strong>注意：</strong>下面这块内容包含的HTML用于在视觉上呈现网页其他位置上的一张数据表格。<a href="#endOfChart">跳过此图表</a></p>')
  .prependTo(canvasContain);
```


可以看到，我们添加了一个“跳过”连接，它引用了一个id属性为endOfChart的元素。为了让这个跳过链接能起作用，我们会创建对应的元素，将它附加到图表容器末尾。我们还会把这个元素的tabindex属性设为-1，这就让它能够通过程序获得焦点，防止Internet Explorer里的一个bug破坏跳跃时的页面滚动。

```
$('<div id="endOfChart" tabindex="-1"/>').appendTo(canvasContain);
```

现在，我们会对视力正常的用户隐藏这段消息，因为它只和那些使用屏幕阅读器浏览的用户有关：

```
.chart-access-message {
  position: absolute;
  left: -99999px;
}
```

快速添加这些代码到图表样式和脚本中后，屏幕阅读器用户现在会将图表识别为一张图像，或者会收到该内容是视觉专用内容的警告，以及一种跳过它的方法。

12.4　让 canvas 图表更进一步：visualize.js 插件

我们把本章讨论的高级概念扩展成一个名为visualize的jQuery插件。这个插件的功能比折线图范例更为全面，它囊括了额外的图表类型，比如柱状图、面积图和饼图，还包含了两套视觉主题样式（除了之前显示的浅白色背景样式，还有一种即将在下方图表里展示的深色图表主题）。此外，它还提供了一套清晰的API，可以用丰富的配置选项（比如宽度、高度、颜色和解析方向等）生成图表。可以用它生成本章里描述的这张图表，方法很简单，只需链接jQuery库和这个插件脚本，然后在网页里的任何表格上调用`visualize`方法即可。

要在某张表格上调用visualize.js插件，请把图表的`type`设置为`line`、`area`、`bar`或`pie`。如果你没有指定一种图表类型，那么默认会创建出一张折线图，如图12-16所示。

```
$('table').visualize({type: 'line'});
```

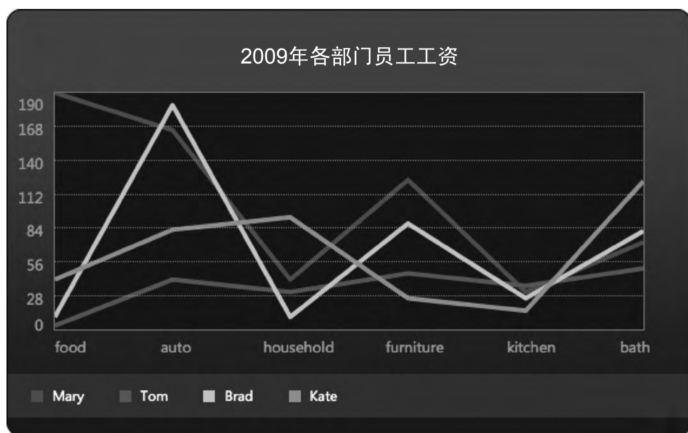


图12-16　通过visualize.js插件创建的折线图

要创建一张每条线下方都带有半透明填充区域的面积图，只需为图表`type`传入`area`即可，如图12-17所示。

```
$('table').visualize({type: 'area'});
```

柱状图的图表`type`则是`bar`，如图12-18所示。

```
$('table').visualize({type: 'bar'});
```

也许和你预计的一样，可通过指定`pie`这个图表`type`用一张饼图显示数据。用visualize.js创建的饼图会自动限制自身的尺寸以适应矩形的`canvas`。这样的尺寸很适合柱状图和折线图，但是对圆形的饼图来说就有点让人感觉拘束。可通过传递一个可选的`height`参数给它一点额外的空间，这样`canvas`就能变得更高一些，饼图也会随之放大，如图12-19所示。

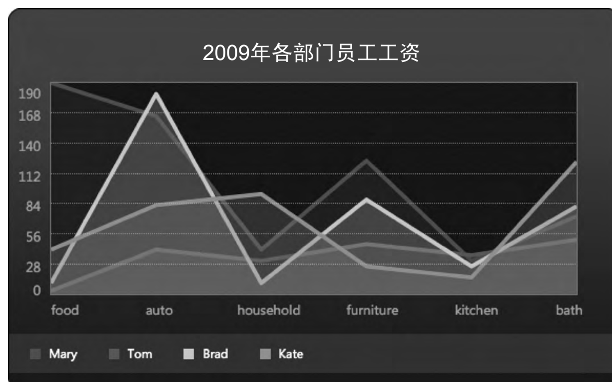


图12-17　通过visualize.js插件创建的面积图

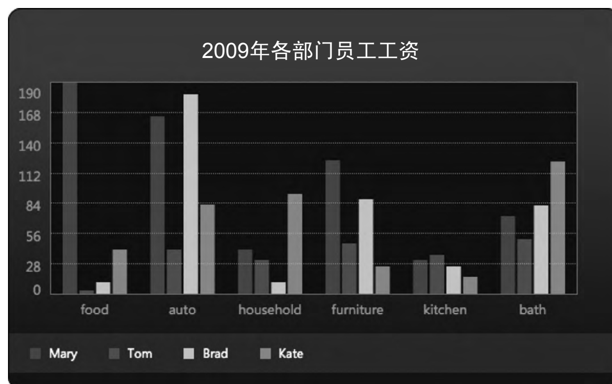


图12-18　通过visualize.js插件创建的柱状图

```
$('table').visualize({type: 'pie', height: '300px'});
```

visualize方法会返回一个新的图表容器div，并自动将它插入到网页中，紧跟在表格之后。要为一张表格显示出多个可视化对象，只需多次调用visualize方法：

```
$(function(){
  $('table').visualize({type: 'pie', height: '300px'});
  $('table').visualize({type: 'bar'});
  $('table').visualize({type: 'area'});
  $('table').visualize({type: 'line'});
});
```

注意　这一节引用的visualize.js插件可以在www.filamentgroup.com/dwpe上下载。要了解关于如何使用visualize.js插件的更多信息，请阅读我们的文章：http://t.cn/zRlT0Lj。

12

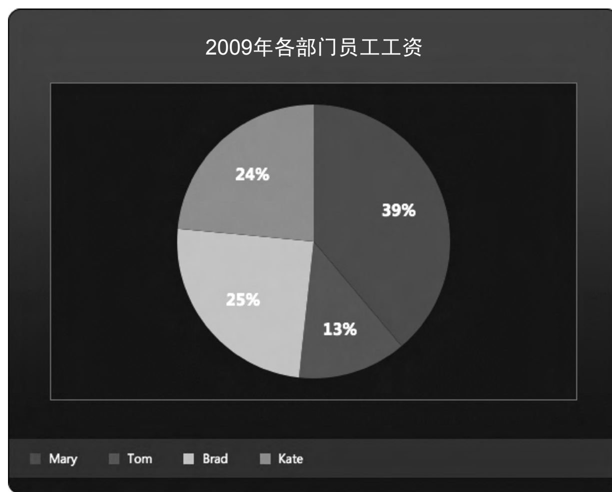


图12-19　通过visualize.js插件创建的饼图

canvas元素提供了一套丰富的API，它们具备健壮的绘图能力，广泛兼容各种浏览器和设备，并且还具备支持使用页面内数据这一优点（它让可访问性得以最大化）。在项目里使用canvas时，只需注意少数几件事即可。

- ▶ 从结构良好的语义数据表格起步，编码进去的意义越多越好。
- ▶ 在引用canvas的同时别忘了引用exCanvas库，以确保Internet Explorer用户能看到你的图表。
- ▶ 用HTML的width和Height属性设置canvas的大小，不能靠CSS来设置尺寸。
- ▶ 如果图表的坐标需要从底部开始（像我们的图表一样），而不是从canvas原生的左上角开始，请使用translate方法设置上下文环境。
- ▶ 如果你选择对视力正常用户隐藏数据表格，那就应使用能让它保留在源代码里的方法，以供屏幕阅读器和搜索引擎访问。
- ▶ 作为给屏幕阅读器用户提供的一项服务，请提供一个“跳过”链接，让他们可以快速越过图表标记，访问网页里那些更有意义的数据。

第 13 章 对话框和叠加层

对话框（Dialog）和叠加层（Overlay）所呈现的内容看上去就像是悬浮在页面上方。对话框通常提供更为丰富的交互内容，比如内嵌的表单元素、按钮和链接，并要求用户做出决定或者进行输入，而叠加层通常呈现的是只读内容。对话框和叠加层的使用范围都很广，包括：

- 呈现系统错误消息和确认信息，要求用户通过交互操作加以关闭；
- 在表单或多步骤向导里收集用户输入信息；
- 点击网页上的某张缩略图后，在一个“灯箱”里显示更大尺寸的照片版本；
- 在文档编辑器等Web应用程序里创建浮动调色板和UI检查器窗口；
- 打开一个“吸附式”的工具提示或者迷你叠加层，提供紧凑格式的内容与功能。

对话框和叠加层都可以采用两种宽泛的交互模型：模式（modal）和非模式（non-modal）：

- 模式对话框或叠加层在打开时会阻止用户与下层网页进行任何交互，限制用户并将用户注意力集中在完成当前的交互操作上。为了明确表现这种关系，许多设计师要求把一个半透明层（颜色或深或浅）放在对话框和网页主体之间，使网页看上去暂时不可用，并且在视觉上帮助突出叠加层。模式对话框和叠加层适用于确认用户的交互操作，提供重要的错误或系统消息、用表单收集用户输入信息，或者只是将注意力集中到特定内容上。
- 非模式对话框或叠加层在打开时能让用户继续与基底网页交互。它适用于对话框根据与用户的交互动态更新网页这种情形，比如浮动调色板、帮助内容面板、搜索框或者聊天窗口。

对话框和叠加层可以有固定的大小和位置，也可以是可移动和可缩放的，以帮助它们营造出桌面PC上工具面板和应用程序窗口的感觉。这两者都可以相当容易地使用现代Web技术构建，但它们可能会给屏幕阅读器带来不便，除非内建了键盘访问、焦点管理和ARIA属性等可访问性功能。另一个重要的考虑事项是如何让内容和功能对基本体验里的用户可用，特别是当你在增强体验里使用Ajax获取叠加层内容时。

这一章会讨论如何用渐进增强方法制作可访问的对话框与叠加层。我们会重点关注对话框，因为你在更简单的叠加层中需要应对的一切复杂状况对话框里都有。我们的范例会使用Ajax来更新对话框内容，从而强调在应对这种更为复杂的情形时，需要使用的特定技巧。

13.1　X 光透视

我们的目标设计是一种经常会用到对话框的情况：某个网站的全局页头上有登录和注册的链接，它们会在网页上层打开迷你表单，让用户留在当时的网站位置。出于本次讨论的目的，我们会把重点放在登录表单上，如图13-1所示，不过注册表单也会使用相同的原则和编程方法。

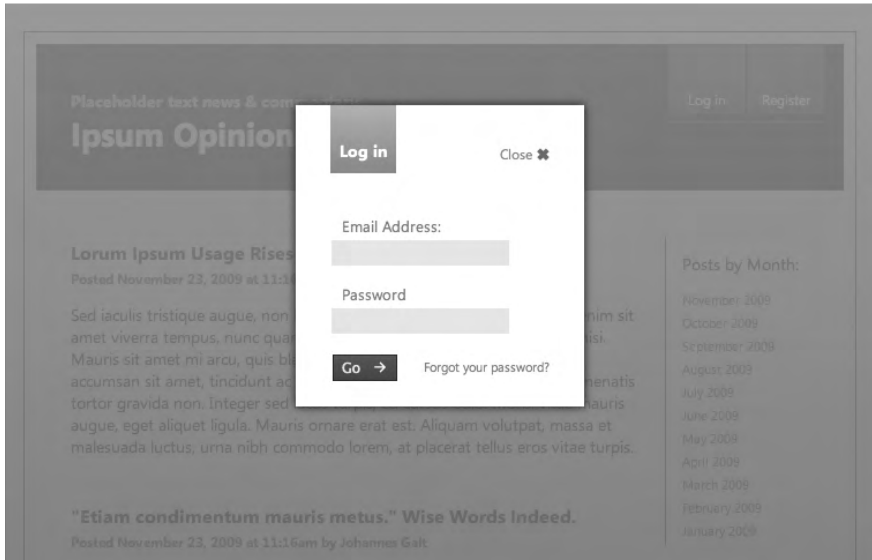


图13-1　登录模式对话框的目标设计

用X光透视这个登录对话框设计，我们能清楚地看出这个对话框是模式的，而且有着固定的大小。它里面的表单字段可以很容易地编写成一张简单的HTML表单，在任何设备上都能工作。最大的问题是：表单代码应该放在标记里的什么位置？

我们有两个选择：把登录表单添加到网站每一张网页的基础标记里，或者链接到一张单独的登录页上。

如果我们把登录表单放在基础标记里，顶部的登录链接就会是一个内部锚链接，点击后会将用户滚动到页面上的标签块。这种方法的优点是内容立即可用并且完全可访问，缺点是这张表单的标记需要放在网站的每一张网页中，从而增加了基本体验里的网页体积和视觉杂乱感，尤其是同时放置登录和注册表单时。

另一个选择是创建一张单独的HTML登录表单页，并在基础标记里链接到这张网页。这样做能最大程度地减小基本体验里的网页体积和视觉杂乱感。

无论哪一种方法都是可行的。最终选择取决于目标网站里对话框的重要性和使用频率，以及各个对话框里的内容标记大小。在当前这种情形里，我们想要尽可能地保持网页轻量化，所以使用单独的链接页这一方式。

在基本体验里，全局页头里的“登录”（Login）链接会将用户导航到一张单独的登录页上。如果用户点击了这张登录页里的“忘记了你的密码？”（forgot your password?）链接，他们就会被导航到一张单独的密码取回页上。如果用户没有正确填写其中任何一张表单，服务器就会重新加载该表单，高亮显示没有正确填写的字段并显示相应的错误消息。如果用户成功填完其中任何一张表单，那么他们就会回到之前所在的页面，而页头会进行更新以反映出他们的登录状态。这些独立链接的网页将完全可以用键盘访问和导航，只要我们为链接和表单使用的是语义化HTML标记，如图13-2所示。

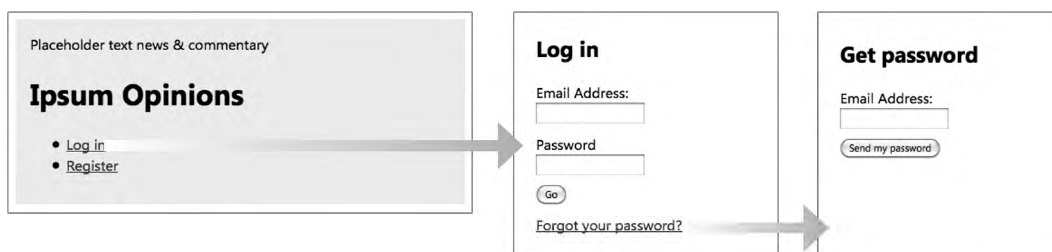


图13-2 基本体验登录顺序里的多张链接网页

基本体验里需要的所有由服务器生成的网页和逻辑都在增强体验里重用。

原生的对话框HTML元素并不存在，增强体验将会用一个`div`作为表单内容的容器，然后在对话框里用Ajax载入和显示它，如图13-3所示。

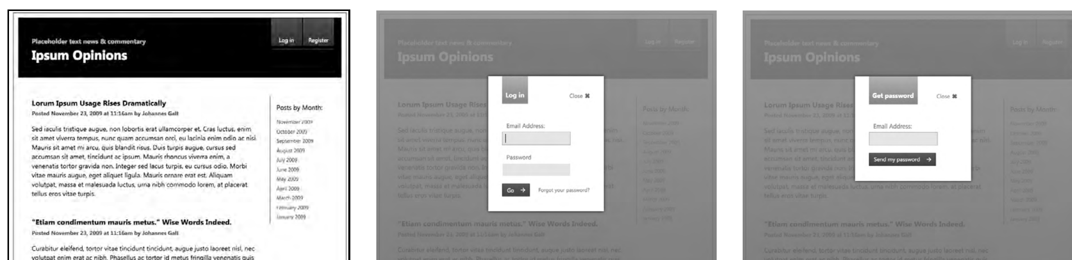


图13-3 增强体验利用Ajax把登录顺序引入对话框

13.2 创建对话框

我们已经勾勒出一种制作登录对话框的方法，接下来就可以编写基本屏幕所需的基础标记，以及用来创建增强模式对话框这一表现方式的增强样式和脚本。

13.2.1 基础标记和样式

我们的基础标记由两部分组成：一个是基础网页里的标准锚链接，它指向包含登录表单的login.html页：

```
<a href="login.html" id="option-login">Log in</a>
```

另一个是一张完整的HTML网页：login.html。它包含了网站的全局导航元素和所有的登录表单内容。

在增强体验里，只需要login.html网页标记中的一个子集，就是包含标题和表单的部分。我们会用Ajax请求完整的网页，然后过滤它的内容，留下想要显示在增强对话框里的那部分内容。

为了给Ajax请求准备这个内容子集，我们会在一个id为login的div容器里存放它，然后创建一个描述网页内容的标题，一张包含多对labels和文本input的登录表单，以及一个提交表单数据的“前进”（Go）按钮。表单元素的后面有一个“忘记了你的密码？”链接，它指向一张单独的密码取回页：

```
<div id="login">
  <h1>Log in</h1>
  <form action="login.php" method="post" id="login-form">
      <label for="email">Email Address:</label>
      <input type="text" class="type-text" name="email" id="email" />
      <label for="email">Password</label>
      <input type="password" class="type-text" name="password"
      id="password" />
      <input type="submit" id="submit-login" name="submit-login" value="Go" />
      <a href="forgot_pass.html" class="alt-option">Forgot your password?</a>
  </form>
</div>
```

这张表单的基础标记在没有应用CSS时也是功能完备的，但它的标签、输入框、按钮和链接都分布在同一行，看上去有些笨拙，如图13-4所示。

图13-4　login.html里的无样式基础标记

为了改善这种状况，我们会给基本样式表添加一些安全样式，包括设置网页的默认字体，以及把label和input元素设为display:block，使它们各占一行。输入框元素则会带有1em的底部外边距样式，从而给表单字段之间提供更多的垂直空间：

```
body, input { font-family: "Segoe UI", Arial, sans-serif; }
label, input { display:block; }
input { margin-bottom:1em; }
```

这些简单又安全的CSS规则让我们的表单在基本体验里更加清晰和易于阅读，如图13-5所示。



图13-5　应用安全样式后的基础标记

13.2.2　增强标记和样式

在增强体验里，我们希望用户的注意力集中在完成登录操作上，不想支持他们同时与基础网页进行交互。为此，要分别为两个组件创建标记：一个是半透明的遮罩，它覆盖基础网页的内容并禁用其功能；另一个是对话框容器，它浮动在遮罩上方，展现表单内容。

要实现这种模式遮罩效果，增强脚本需要在body元素末尾的增强标记里插入两个元素：一个是获取背景色和透明度变化的div，另一个是它内嵌的iframe，后者施加的保护行为能防止用户与下方的网页进行交互。必须设置这个iframe的src，否则一些浏览器会发出安全警告，所以我们会把它设为javascript:false;，起空值的作用，防止它发起不必要的HTTP请求：

```
<div class="modal-screen"><iframe src="javascript:false;"></iframe></div>
```

div的高度和宽度样式被设置为100%以填满整个屏幕，它的深灰色渐变背景图像具有90%的opacity，让网页内容在支持CSS透明度的浏览器里能透过遮罩层略微显示一点。它还有一个很大的z-index值，以确保它会堆叠在其他设置了z-index属性的网页元素之前：

```
.modal-screen {
  background:#717174 url(../images/bg-modal-screen.png) top repeat-x;
  position:absolute;
  width:100%;
  overflow:hidden;
  height:100%;
  top:0;
  left:0;
  opacity:.9;
  z-index: 9999;
}
```

这个div里的iframe是作为一种变通方式来解决某些浏览器里的z-index（即CSS堆叠顺序）问题，这个问题会导致浏览器错误地将网页元素（包括下拉菜单和带有滚动条的元素）显示在遮

13

罩层和对话框的前面，无论样式表里如何设置z-index堆叠顺序都是如此。iframe则能够一致而可靠地堆叠在任何网页元素的上方，因此它就像是模式遮罩层和对话框的一个保护层。我们会把它的高度和宽度设为100%来填满整个模式遮罩层div，opacity则设置为0（我们会用JavaScript强化这个属性，让它在原生不支持CSS opacity属性的Internet Explorer 6里也能正常工作）。虽然它是透明的，但是它仍然能避免任何z-index问题的出现。

```
.modal-screen iframe {
  height:100%;
   width:100%;
   position:absolute;
   top:0;
   left:0;
   padding:0;
   margin:0;
   opacity: 0;
```

为了完成遮罩层效果，脚本会在对话框显示时给body元素添加一个class，去除浏览器窗口里的所有滚动条，防止用户将对话框滚动出视野，如图13-6所示。

```
body.blocked {
  overflow:hidden;
}
```

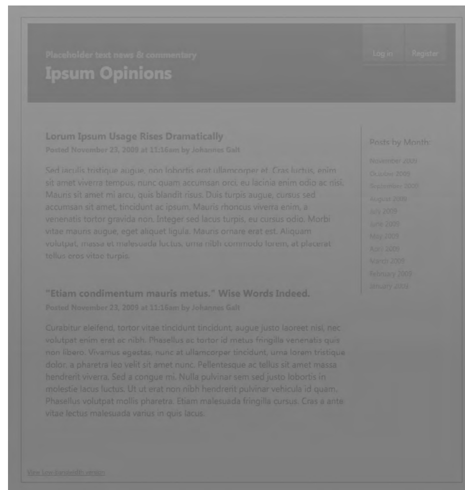


图13-6　模式遮罩层覆盖了网页的全部内容

草拟出模式遮罩层的标记和样式后，现在可以继续编写对话框的代码了。首先，给body元素添加一个值为application的role，使屏幕阅读器能正确识别这个对话框组件，并让我们能编写自定义的键盘行为：

```
<body role="application">
```

为了给对话框创建一个容器，增强脚本会生成一个ID为dialog的div：

```
<div id="dialog"></div>
```

为了支持可访问性，给这个div添加一个值为dialog的ARIA role属性，告诉屏幕阅读器它不是一个标准的div，而是在扮演对话框UI组件这一角色。我们还会给这个div指派一个唯一的id，它会被“关闭”（Close）链接引用。另外，设置一个值为false的aria-hidden属性，告诉屏幕阅读器此对话框当前是可见的：

```
<div id="dialog" role="dialog" aria-hidden="false"></div>
```

增强脚本会用Ajax载入整张login.html页，找到基本页面的登录表单内容div（即#login）里的所有内容，然后将这块标记添加到对话框div里：

```
<div id="dialog" role="dialog" aria-hidden="false">
  <div id="login">
   <h1>Log in</h1>
   <form action="login.php" method="post">
   <label for="email">Email Address:</label>
   <input type="text" class="type-text" name="email" id="name" />
   <label for="email">Password</label>
   <input type="password" class="type-text" name="password"
   id="password" />
   <input type="submit" id="submit-login" name="submit-login" value="Go" />
   <a href="forgot_pass.html" class="alt-option">Forgot your password?</a>
  </form>
  </div>
</div>
```

考虑到样式的一致性，脚本会用一个带有dialog-content类的容器div围住对话框的全部内容：

```
<div id="dialog" role="dialog"aria-hidden="false">
  <div class="dialog-content">
      <div id="login">
          <h1>Log in</h1>
          ...login form markup...
      </div>
  </div>
</div>
```

根据ARIA规范的规定，使用dialog这个role时，应该在标记里添加aria-labelledby属性，指向扮演对话框标签角色的元素id。为了遵循这一规定，增强脚本会给对话框内容里的h1元素指派一个dialog-title的id，以供这个属性引用。这一联系建立后，屏幕阅读器就会知道“登录”标题描述了这个对话框的用途，并可能会在对话框出现时朗读出“登录对话框”。

```
<div id="dialog" role="dialog" aria-hidden="false"aria-labelledby= "dialog-title" >
  <div class="dialog-content">
      <div id="login">
          <h1  id="dialog-title" >Log In</h1>
          ...login form markup...
```

13

```
      </div>
    </div>
  </div>
```

注意　如果标记里没有这个属性能够关联的描述性元素（比如一个标题），可以使用aria-label属性，把它作为对话框标题提供给屏幕阅读器。

增强脚本会在标题元素后面附加一个链接，它具有点击后关闭对话框的行为。可以告诉屏幕阅读器这个链接扮演了“关闭”按钮的角色，方法是给这个链接指派一个ARIA属性aria-controls来引用对话框div的ID，并添加role=button属性：

```
<div id="dialog" role="dialog" aria-hidden="false"
 aria-labelledby="dialog-title">
   <a href="#" class="dialog-close" role="button" aria-controls="dialog"> Close</a>
   <div class="dialog-content">
       <div id="login">
           <h1 id="dialog-title">Log In</h1>
           ...login form markup...
       </div>
    </div>
 </div>
```

对话框div的增强CSS会将它设置成固定宽度，并将它绝对定位到屏幕中央。我们的做法是设置一个50%的left值，然后用等同于半个对话框宽度（即125px）的左侧外边距负值进行偏移。我们会在支持CSS3盒阴影属性的现代浏览器里添加阴影效果，并设置一个z-index把它移到模式叠加层的前面：

```
#dialog {
  background:#fff;
  position:absolute;
  width:250px;
  top:20%;
  left:50%;
  margin-left:-125px;
  -o-box-shadow:0 0 8px #111;
  -moz-box-shadow:0 0 8px #111;
  -webkit-box-shadow:0 0 8px #111;
  box-shadow:0 0 8px #111;
  z-index: 10000;
}
```

我们把标题和“关闭”按钮都浮动到了左侧，让它们能够并排显示。标题有一张背景渐变图像和一些样式，这些样式的作用是给文本设置让人感觉舒服的外边距和内边距。“关闭”链接有一个“x”图标，通过在文本左侧添加一张背景图像来实现，并加入了鼠标悬停时显示下划线的样式规则：

```
#dialog-title {
  background:#8c8c8d url(../images/bg-dialog-title.png) top repeat-x;
  font-size:1.4em;
  color:#fff;
  margin:0 0 20px 30px;
  padding:30px 8px 10px;
  float:left;
  text-decoration:none;
  outline:none;
}
.dialog-close {
  position:absolute;
  top:34px;
  right:30px;
  color:#888;
  font-size:1.2em;
  text-decoration:none;
  background:url(../images/icon-close.png) right 50% no-repeat;
  padding-right:15px;
}
.dialog-close:hover {
  text-decoration:underline;
}
```

包围对话框内容的div清除了标题和“关闭”链接的浮动，这样它就会总是处于两者的下方。form元素的样式设置了基本的字体格式、间距，以及标签、文本输入框和“提交”（Submit）按钮所需的颜色：

```
.dialog-content {
  clear: both;
}
form {
  padding:10px 30px 20px;
  overflow:auto;
  clear:both;
}
form label {
  display:block;
  font-size:1.3em;
  color:#808080;
  margin:.6em 0 .3em;
  padding-left:10px;
}
form input.type-text {
  margin:.3em 0 1.2em;
  display:block;
  border:1px solid #fff;
  background:#E6E6E6;
  padding:.3em .4em;
  font-size:1.3em;
}
```

:focus伪类让我们可以在用户把焦点移到文本input上时，为其指定更深的边框颜色。一些

浏览器会给获得焦点的元素添加额外的视觉反馈，比如苹果公司的Safari浏览器中会出现蓝色的光，如图13-7所示。虽然可以通过设置输入框的CSS outline属性来覆盖这种反馈，但我们的建议是：只要有可能，就让各种浏览器按照它内部统一的方式处理外框焦点的样式：

```
form input.type-text:focus {
  border-color:#aaa;
}
```



图13-7　添加样式后的增强体验登录表单标记

13.2.3　对话框增强脚本

我们已经创建了基础标记，并规划了对话框所需的增强标记和样式，现在编写JavaScript来生成对话框的标记，并给那些通过能力测试的浏览器指派行为。

首先，给body元素添加值为application的role：

```
$('body').attr('role','application');
```

只要点击增强体验里的"登录"链接就会打开登录对话框。为此，首先给链接绑定一个点击事件：

```
$('#option-login').click(function(){
  //……这里放点击事件的脚本
});
```

1. 生成来自Ajax的增强标记

只要点击这个链接，脚本就会用Ajax向Web服务器请求对话框的内容。我们会使用jQuery的$.get工具方法（utility method）借助Ajax载入网页，方法是传递登录链接的href属性作为我们想要请求的URL：

```
$('#option-login').click(function(){
  $.get( $(this).attr('href') );
});
```

$.get这个Ajax工具提供了一个回调函数，它会在服务器响应并返回login.html页时执行。这个响应（由下面的response变量提供）是HTML网页源代码的全文，因此需要在其中搜索并找到我们要的特定对话框标记。dialogContent变量从完整的Ajax响应里引用了登录页中的登录div元素：

```
$('#option-login').click(function(){
  $.get($(this).attr('href'), function(response){
      //获取ajax的响应文本，在一个$()里引用它
      var response = $(response);
     //找到响应文本里的登录div，用于对话框内容
     var dialogContent = response.find('#login');
  });
});
```

有了这些内容之后，脚本就可以根据我们之前规划的增强标记模板生成对话框：

```
$('#option-login').click(function(){
  $.get($(this).attr('href'), function(response){
      // 获取ajax的响应文本，在一个$()里引用它
      var response = $(response);

      // 找到响应文本里的登录div，用于对话框内容
      var dialogContent = response.find('#login');

      // 找到将会扮演对话框标题角色的元素
      var dialogTitle = dialogContent.find('h1');

      //给标题添加ID属性，以供ARIA引用
      dialogTitle.attr('id', 'dialog-title');

      //创建模式遮罩层
      var modalScreen = $('<div class="modal-screen">
      ➥<iframe src="javascript:false;"></iframe></div>');

      //设置模式遮罩层的透明度以修补Internet Explorer
      modalScreen.children(0).css('opacity',0);

      // 创建对话框的包装器div
      var dialog = $('<div id="dialog" role="dialog"
      ➥aria-hidden="false" aria-labelledby="dialog-title"></div>');

      // 创建关闭链接
      var close = $('<a href="#" class="dialog-close"
      ➥role="button" aria-controls="dialog">Close</a>')
          .appendTo(dialog);

      // 创建内容div
      var content = $('<div class="dialog-content"></div>')
          .append(dialogContent)
          .appendTo(dialog);
  });
});
```

在这个阶段，脚本已经用Ajax获取了完整的登录页，解析出了表单所需的相关内容并且给叠加遮罩层和对话框收集了完整的增强标记，它已经准备好给对话框标记应用事件了。

2. 给增强标记应用行为

为了用脚本给对话框标记应用事件，首先创建一个自定义的close事件，它列出了对话框关闭时必须发生的所有事情：移除模式遮罩层、对话框标记，以及body上用来防止窗口滚动的blocked类。这个事件可以在多种情形下触发：点击“关闭”按钮、按下Esc键或者登录成功。考虑到上下文环境和清晰度，后面的代码范例会用省略号来代替它们先前的部分代码。

```
$('#option-login').click(function(){
  $.get($(this).attr('href'), function(response){
      ……先前的对话框代码……
      //给对话框div绑定一个自定义关闭事件
      dialog.bind('close',function(){
         //移除页面里的模式遮罩层
         modalScreen.remove();

        //移除页面里的对话框div
        dialog.remove();

       //移除body上的blocked类
       $('body').removeClass('blocked');
    });
  });
});
```

我们会在点击“关闭”按钮时触发close事件，并让click事件返回false来防止URL井号串更新为链接的href值：

```
$('#option-login').click(function(){
  $.get($(this).attr('href'), function(response){
       ……先前的对话框代码……
       //给关闭按钮绑定click事件
       close.click(function(){
            //触发对话框的close事件
            dialog.trigger('close');

            //防止链接的href #修改url
            return false;
       });
  });
});
```

close事件还会在用户每次按下Esc键时触发，方法是寻找keydown事件里按键的keyCode（27）：

```
$('#option-login').click(function(){
  $.get($(this).attr('href'), function(response){
      ……先前的对话框代码……

      $(document).keydown(function(ev){
          if(ev.which == 27){
```

```
            dialog.trigger('close');
          }
        });
      });
    });
```

3. 添加和显示对话框

生成对话框标记并绑定事件之后，增强脚本会给body元素添加blocked类来防止它滚动，并把完成后的对话框组件插入网页，使其可见：

```
$('#option-login').click(function(){
  $.get($(this).attr('href'), function(response){
      ……先前的对话框代码……
      //找到body元素，添加类，插入对话框
      $('body')
        .addClass('blocked')
        .append(modalScreen)
        .append(dialog);
   });
});
```

4. 管理焦点

通过管理键盘焦点来优化导航和可访问性是构建快速高效对话框的一个关键步骤。ARIA规范建议把光标焦点移到对话框内，这样用户就可以开始和里面的内容交互了。否则的话，如果对话框打开后焦点还停留在下方的网页上，就会让人感到非常困惑。

在我们的登录表单中，“关闭”链接虽然从技术上看是对话框里第一个可获得焦点的元素，但是，对话框打开时最有用的可获得焦点元素却是第一个文本输入框，因为用户可以立即开始输入他们的邮箱地址。我们会用jQuery找到dialogContent里的第一个input，然后把焦点移到它这里：

```
$('#option-login').click(function(){
  $.get($(this).attr('href'), function(response){
      ……先前的对话框代码……
      //聚焦登录表单里的第一个input
      dialogContent.find('input')[0].focus();
  });
});
```

当焦点放在第一个文本input上后，用户就可以用Tab键导航到接下来的密码input、提交按钮，直至最后的“忘记了你的密码”链接。

再次按下Tab键产生的原生行为是将焦点移出对话框，转到底下网页源代码顺序里下一个可获得焦点的项上。这会让用户感到困惑，因为对话框被设计成模式对话框。脚本会通过程序把焦点限制在对话框内，从“关闭”链接开始，然后移向各个表单字段。我们实现这种限制的方法是给整个document绑定一个focus事件，只要焦点发生变化就会触发该事件。变化发生时，脚本会检查新近获得焦点的元素是否在对话框div之外，如果是，就把焦点移动到对话框内第一个可获得焦点的元素上：

```
$('#option-login').click(function(){
  $.get($(this).attr('href'), function(response){
      ……先前的对话框代码……

    //原生可获得焦点的元素列表，加上那些带有tabindex的元素
    var focusable = 'a,input,button,textarea,select,[tabindex]';

        //给document绑定一个focus事件
        $(document).bind('focus', function(ev){

            //如果网页上有对话框，并且获得焦点的元素不在该对话框内
            if($('#dialog').length &&
                !$(ev.target).parents('#dialog').length){
                //把焦点移到对话框里第一个可获得焦点的元素上
                dialog.find(focusable)[0].focus();
            }
        });
    });
});
```

13.3 让对话框更进一步

前面的摘要涵盖了创建单个固定大小的模式对话框所需的一切标记、样式和脚本范例，但是你可能想实现许多其他对话框/叠加层功能与方案，包括用非模式对话框来提供工具面板，制作像照片幻灯片那样显示图片的简单叠加层，以及用可缩放的帮助文本叠加层在页面上方显示补充的内容。

jQuery UI对话框插件包含了一套扩展功能集来实现这些对话框和叠加层方案（例如模式和非模式的相应能力、缩放、拖动、定位，以及对同时在网页上显示多个对话框的支持能力），还考虑到了ARIA支持。不仅如此，jQuery UI对话框还可用ThemeRoller工具修改样式。可以到这里访问jQuery UI对话框插件的演示和文档：jqueryui.com/demos/dialog，下载这个对话框插件的地址是：jqueryui.com/download。

13.4 使用对话框脚本

本书附带的演示和代码（位于本书的网站上：www.filamentgroup.com/dwpe）包含一个脚本：jQuery.dialog.js。它将本章展示的创建单个对话框的技巧集中到一个jQuery插件里，便于你在你的项目里使用。

要在你的网页里使用这个脚本，请下载并引用对话框范例页里列出的那些文件，然后只需找到网页上的某个元素（甚至可以是jQuery函数封装的一串HTML标记）并对其调用`dialog`方法即可。举个例子，假设网页上有一张id为`login`的表单，要把这张表单显示在一个对话框里，需要用一个jQuery选择器找到它并对其调用`dialog`方法：

```
$('form#login').dialog();
```

dialog方法会自动生成一个对话框容器，用于存放本章列举的所有标记和行为，例如模式保护遮罩层和“关闭”链接。调用这个方法时，它首先会关闭网页上现存的所有对话框，确保任一时刻只有一个对话框是可见的。dialog方法还包含了title、buttons和focus这些可配置的选项，可以用一个包含键/值对的对象来传递它们。

title选项用来将对话框里的某个特定元素指定为提供给屏幕阅读器用户的标题，插件会自动给这个元素应用一个dialog-title的id。title选项接受一个jQuery选择器作为它的值，如果没有指定title选项，插件会默认使用第一次出现的下列元素之一：h1、h2、h3、h4、h5、h6、legend、label和p。举个例子，下面的代码将title选项设置为对话框内容区域里的第一个span元素：

```
$('form#login').dialog({
  title: 'span'
});
```

buttons选项让你能生成将出现在对话框底部的按钮。这在某些情形下很有帮助，比如当对话框扮演提示消息的角色，要求用户必须进行确认和取消时。buttons选项接受它自己的键/值对对象，为这些对分别生成button元素，并将它们添加到一个带有dialog-footer类的div里，这个div则会放置在对话框标记的末尾。在这些键/值对中，键作为按钮自身的HTML文本，它的值则是一个会在点击按钮时执行的函数。举个例子，下面的脚本生成一个带有两个按钮的对话框，点击这些按钮时会分别触发不同的警告：

```
$('div#documentUnsavedPrompt').dialog({
   buttons: {
       'Okay': function(){
           alert('你点击了确定');
       },
       'Cancel': function(){
           alert('你点击了取消');
       },
  }
});
```


focus选项让你指定哪个特定元素应该在对话框打开时获得焦点。默认情况下，插件会尝试聚焦对话框内容div里第一个原生可获得焦点的元素（比如某个锚、表单元素或者任何带有tabindex属性的元素）。如果找不到这样的元素，它会选择任何由buttons选项生成的按钮作为备用方案，最后才会选择对话框的“关闭”链接。可以指定对话框内某个特定元素的jQuery选择器来覆盖focus选项的默认设置。这在类似提示确认的情形中也许很有用，因为你可能想把焦点移到“确定”按钮上，忽略对话框里的内容：

```
$('div#documentUnsavedPrompt').dialog({
  buttons: {
      'Okay': function(){
          alert('你点击了确定');
      },
      'Cancel': function(){
          alert('你点击了取消');
      },
```

```
  },
  focus: '.dialog-footer button:first'
});
```

对话框插件还附带一个工具函数（和dialog方法有着相同的名称），它被设计用来创建源于外部内容的对话框。dialog工具函数需要一个URL参数来指明对话框内容的位置。举个例子，下面的脚本会用login.html页里的内容创建一个对话框：

```
$.dialog('login.html');
```

这个对话框会立即打开，在发送Ajax请求时显示一段“加载中……”的消息。只要请求返回，对话框会用login.html的内容和options参数里指定的设置重建自己。可以配置加载时显示的消息，方法是通过options参数传递任意文本串给loadingText选项：

```
$.dialog('login.html',{
  loadingText: '请稍候……'
});
```

假设login.html是一张完整的网页，包括html、head和body元素，并引用了一些JavaScript和CSS文件，这时你很可能只想让网页内容的一个子集出现在对话框里。dialog工具函数让这一切变得简单：只需在URL路径后添加一个空格，然后附上你想要的所有jQuery选择器即可。举个例子，下面的脚本会用login.html页中某个id为login的div里的内容创建一个对话框：

```
$.dialog('login.html #login');
```

可以使用这种方法轻松制作出本章描述的范例，具体做法是给“登录”链接绑定一个click事件，让它基于链接href属性引用的网页生成一个对话框：

```
//给登录链接指定click事件
$('#option-login').click(function(){
 //用链接href属性所引用网页里的#login div创建对话框

 $.dialog( $(this).attr('href') +' #login' );
 //防止浏览器刷新
 return false;
});
```

dialog工具插件还可以设置一个options参数，后者接受刚才介绍的所有可配置对话框选项（即title、buttons和focus），以及一个额外的complete选项。complete选项接受一个回调函数，此函数会在Ajax响应返回并填入对话框内容后执行。向complete函数传递一个变量参数，就能引用Ajax响应的HTML，然后可以进一步修改它，比如给对话框里的标记添加事件。下面的例子扩展了之前的脚本，它所指定的complete回调函数会挖掘响应内容以寻找“忘记了你的密码”链接，然后绑定一个点击事件来请求它的内容，用于一个新的对话框：

```
//给登录链接指定click事件
$('#option-login').click(function(){
  //用链接href属性所引用网页里的#login div创建对话框
  $.dialog( $(this).attr('href')+' #login', {

      //指定complete回调函数，用响应作为参数
```

```
        complete: function(response){

            //找到忘记密码链接，绑定click事件
            response.find('a.alt-option').click(function(){
            //用链接href属性所引用网页里的#forgot div创建对话框

            $.dialog($(this).attr('href')+' #forgot');
            //防止浏览器刷新
           return false;
        });
      }
   });
   //防止浏览器刷新
   return false;
  });
```

创建可访问的对话框和叠加层十分容易，前提是你要记得加入恰当的ARIA属性并管理焦点，以确保能正确支持键盘。这里列举的基本编码原则同样适用于各种各样的不同组件，从照片灯箱叠加层，到可拖动和缩放的面板、检查器以及调色板不等。

第 14 章

按　　钮

按钮（button）在网站和应用程序界面里无处不在（用来提交数据和在工具栏、表单和导航控件里发起操作），设计师们经常会定制它们的外观。通常情况下，我们为项目设计自定义样式按钮的原因是按钮必须：

- 匹配整个网站或应用程序的品牌风格和表现方式；
- 使用颜色、纹理和尺寸来传达按钮层级，例如从视觉上区分首要和次要操作；
- 通过加入图标使它们更具有独特的可识别性，更易于浏览，或者干脆是紧凑的纯图标按钮，就像文本编辑工具栏里使用的那些；
- 引入自定义的交互状态来提供更好的反馈，比如触摸屏上按钮的按下状态。

只需加一点CSS和JavaScript，几乎所有的HTML元素就都能具备按钮的外观和行为：举个例子，谷歌公司的Gmail网页应用程序在界面里到处使用基于div的按钮。但是，依靠JavaScript事件来处理表单提交是有风险的，脚本功能被禁用或不被完全支持时，所有相关功能就会变得不可用。

为了编写能在所有地方可靠工作的基础标记，需要从type为button的input元素或者button元素起步。选择哪一种元素要根据特定情形下的样式和功能需求而定。

这一章会分步骤实现两种目标设计，讨论在基础标记里选择使用input或button元素的判断标准，然后分析如何在增强体验里把它们转变成格式丰富的按钮。

14.1　X 光透视

假设有一个社交网站，里面的某张网页列出了好友邀请，用户可以接受或忽略它们。我们的设计为这些按钮引入了视觉层级，强调了积极的“添加好友”（Add Friend）操作，而不是消极的忽略操作。考虑两种可能的目标设计，它们列举了一些关键要素，能帮助我们决定该使用哪种原生元素：第一种设计比较简单，它给两个按钮添加了差别化的样式，显示出强调程度的不同，如图14-1所示。

另一种设计更复杂一些，它需要给每个按钮内置一个图标，同时使用混合字形的文本格式——“添加约翰”（Add John）是粗体，如图14-2所示。

可以使用两种原生元素来编写自定义按钮的基础标记：一种是type属性设置为submit、reset或image的input元素，另一种是button元素。在这两种标记选择中，input元素在HTML规范里存

在的时间比button长（1996年左右的HTML3规范收录了input，相比之下button则是在1999年年末的HTML4规范里才被加入），而且它往往能在范围更广的旧版和移动浏览器里正确工作。因为它的普遍可用性，所以input是在基础标记里创建按钮的首选。



图14-1　目标按钮设计带有一种区分主要和次要按钮的视觉样式



图14-2　带有复杂按钮格式的目标设计

某个设计需要用到带有样式的按钮时，设计师通常会避免使用基于input的标准按钮。一种常见的误解是input不能用CSS可靠地添加样式。我们测试了这个说法，惊奇地发现可以在所有主流浏览器（甚至是Internet Explorer 5）上让基于input的按钮具备一致的样式，包括各种字体样式、背景图像、边框、圆化的边角、自定义尺寸、外边距和内边距。

但是，input存在一个关键的限制：这个元素由一个单独的自关闭标签组成，因此它无法容纳别的HTML元素，比如图像、span、em、strong或其他任何HTML标签。这就阻碍了我们把input用于复杂文本格式，层叠多张背景图像，或者添加多个图标。

相比之下，button元素则拥有开始和关闭标签，能够容纳额外的HTML元素。这样我们就有地方存放复杂目标设计里的所有设计特性：可以用一个strong标签围住“添加约翰”来实现强调，然后添加一个span标签来应用图标。但是，button有它自己的缺点：因为它在最早的HTML规范中不存在，所以一些老式浏览器和移动平台不能识别它，从而导致按钮完全无法显示，或者不能在点击按钮后正确提交表单。

因此，建议基础按钮标记总是使用input，只有在增强体验里才使用button元素。对于更简单的第一种设计，我们会在增强体验里给input添加样式，就像对其他任何元素所做的一样；对于带有格式化文本和图标的复杂设计，把input“转换”成一个button来实现高级格式，同时保留原生提交表单数据的能力。

首先，分析如何给基于input的按钮添加样式。然后，分步介绍如何在增强体验里把input“转换”成一个button元素，以容纳第二个例子里的额外图标和文本格式。

14.2　给基于 input 的按钮添加样式

我们会用基于input的按钮构建第一种设计，同时在基本体验和增强体验里用CSS给它添加样式。该设计（如图14-1所示）足够简单，因此这一方法在各种各样的浏览器里都可行。

为了让主要和次要按钮之间存在一种视觉层级，在基础标记里给input元素的标记做出区分，方法是指派描述性的类（btn-primary和btn-secondary）。为了安全起见，我们会把不少视觉样式（比如“添加”按钮上的深色背景浅色文字）留给增强体验，因为旧版的浏览器可能只支持文字颜色样式（而不支持背景图像或者颜色），这可能会导致文字不可阅读。

但是，可以在基本体验里显示出视觉重点，方法是将主要按钮的font-weight设置为粗体。（虽然这个样式属性一般都能得到良好支持，但是有些浏览器可能会忽略它，因为它应用在一个input上。）即使在非常原始的浏览器上，我们给主要操作和替代操作所加的提示也能在基本体验里传达出来（如图14-2所示）。

14.2.1　基础标记和样式

我们这个社交网站的基础标记始于一个form标签。表单里有一个fieldset，它包含了一个legend，一个显示地点和共同好友数量的段落，以及两个用于添加和忽略好友请求的提交input元素：

```
<form action="friendForm.php">
  <fieldset>
      <legend>Friend request from John Smith</legend>
      <p>Boston MA - 16 friends in common with you</p>
      <input type="submit" name="add" value="Add John as a friend" />
      <input type="submit" name="ignore" value="Ignore" />
  </fieldset>
</form>
```

我们会给每一个input元素都添加一个类，标明它的角色是主要还是次要：

```
<input type="submit" name="add" value="Add John as a friend"
class="btn-primary" />

<input type="submit" name="ignore" value="Ignore" class="btn-secondary" />
```

我们的基础标记现在已经能为所有支持基本HTML的设备提供一种可用的体验了，如图14-3所示。

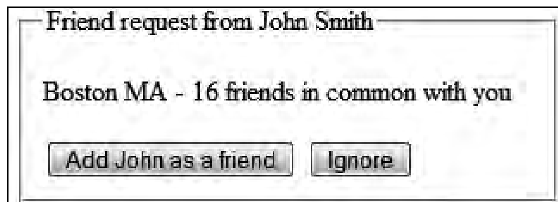


图14-3　不带样式的基础标记

这张表单完美可用，但可以做一些视觉上的打磨，因此给基本样式表添加一些安全CSS规则。

font-family样式属性对外观感受有着巨大影响。网页上的大多数元素从它们的父元素那里继承font-family属性，但表单控件通常是个例外。许多浏览器会为表单控件指定某种等宽字体，忽略网页上设置的其他字体样式。为了修复这个问题，把font-family规则指向body和input元素：

```
body, input { font-family: "Segoe UI", Arial, sans-serif; }
```

这条简单的规则已经让表单变得好看多了！效果如图14-4所示。



图14-4　应用了字体样式的基础标记

为了提高表单的易读性，我们会移除fieldset的边框，调整它的外边距，把legend的样式设成粗体和绿色，并调整里面段落的外边距：

```
fieldset {
  margin-top: 1.2em;
  margin-bottom: 1.2em;
  border: 0;
}

legend {
  color: #339900;
  font-weight: bold;
}

fieldset p {
  margin: 0 0 1.2em;
}
```

现在，基本体验里的表单块看上去更接近我们的目标设计了，如图14-5所示。



图14-5　应用了fieldset、legend和段落样式的基础标记

最后，给主要按钮添加粗体文本样式，并指定次要按钮的文本应该是普通文本（不是粗体），以确保基本体验里存在视觉重点的变化：

```
.btn-primary { font-weight: bold; }
.btn-secondary { font-weight: normal; }
```

这些样式就位后，“添加”按钮就有了视觉上的优先性，如图14-6所示。



图14-6　基础标记里的主要按钮带有粗体文字，用于突出视觉重点

完成基础标记和样式之后，我们将要处理增强标记和样式，它们会在浏览器通过能力测试时应用到网页上。

14.2.2　增强标记和样式

在我们的目标设计里，两个按钮具备许多相同的样式，因此在增强样式表里为这两个按钮选择器编写同一条规则，以应用这些共享的样式。这些规则定义了内边距、字体格式和对齐方式，并把input上的光标样式设为可点击的手形图标：

```
 .btn-primary,
 .btn-secondary {
   padding: .4em 1em;
   border: 1px solid #aaa;
   background-color: #eee;
   text-align: center;
   cursor: pointer;
   font-size: 1.2em;
}
```

注意　这些按钮的字体粗细会继承之前写入基本样式表的主要和次要按钮类，所以不需要在增强样式表里再次定义它们，如图14-7所示。



图14-7　应用了共同样式的input元素

提示　设置按钮样式时，要为内边距和外边距这类属性使用em单位，而不是像素，以确保按钮（和它占据的空间）会根据字体的大小伸展和收缩。

老版本的Internet Explorer会给input和button元素增加额外的宽度，而且不能用padding或margin的值来改变，不过有一种方法可以修复这个问题：指定一个width属性，并将overflow属性设为visible。如果按钮不要求有具体的width，可以将它的值设为auto：

```
.btn-primary,
.btn-secondary {
  padding: .4em 1em;
  border: 1px solid #aaa;
  background-color: #eee;
  text-align: center;
  cursor: pointer;
  font-size: 1.2em;
  width: auto;
  overflow: visible;
}
```

为了圆化按钮的边角，应用CSS3的border-radius属性，如图14-8所示。写作本书时，只有基于Mozilla和Webkit的浏览器支持border-radius属性（通过使用各自专用的属性名），但是使用它仍然是安全的，因为不支持这个属性的浏览器会显示方角按钮。我们还会加上W3C规定的标准border-radius属性，这样的话，未来实现了这一特性的浏览器也能够显示出圆角（写作本书时，Opera和Internet Explorer都已致力于支持这个属性[①]）：

```
.btn-primary,
.btn-secondary {
  padding: .4em 1em;
  border: 1px solid #aaa;
  background-color: #eee;
  text-align: center;
  cursor: pointer;
  font-size: 1.2em;
  width: auto;
  overflow: visible;
  -moz-border-radius: 7px;
  -webkit-border-radius: 7px;
  border-radius: 7px;
}
```




图14-8 带圆角的基本按钮样式

基本样式里的主要和次要按钮类指定了背景图像（分别是绿色和银色），以及匹配目标设计的文本和边框颜色：

```
.btn-primary {
  background: #459e00 url(../images/button-green.gif) no-repeat left center;
  color: #fff;
```

① Opera 10.5、IE 9、Safari 5、Chrome、Firefox 4、iOS 4和Android 2.1都已经支持标准的border-radius属性。
——译者注

```
  border-color: #2d7406;
}

.btn-secondary {
  background: #e7e8e9 url(../images/button-silver.gif) no-repeat left center;
  color: #555;
  border-color: #b3b3b3;
}
```

为使文字在背景图像里显得突出，用`text-shadow`属性给按钮文字添加了阴影，如图14-9所示。像`border-radius`属性一样，写作本书时只有少数几种浏览器能支持`text-shadow`（具体来说是Firefox 3.5+、Safari 1.1+和Opera 9.5+），不过加入这个属性是无害的，只要文字在没有阴影时有良好的对比度就行。

```
.btn-primary {
  background: #459e00 url(../images/button-green.gif) no-repeat left center;
  color: #eee;
  border-color: #2d7406;
  text-shadow: -1px -1px 0 #37730e;
}

.btn-secondary {
  background: #e7e8e9 url(../images/button-silver.gif) no-repeat left center;
  color: #555;
  border-color: #b3b3b3;
  text-shadow: 1px 1px 0 #fafafa;
}
```



图14-9　应用按钮样式后

为了在用户和按钮交互时提供良好的视觉反馈，为每种按钮类型各定义了一个悬停状态，如图14-10所示。理想情况下，使用`:hover`伪类，但是Internet Explorer 6和之前的版本只在锚元素（`a`）上支持这个伪类。为了解决这个问题，当用户将光标移上和移开按钮时，分别用JavaScript添加和移除一个悬停类：

```
.btn-primary-hover {
  background-color: #57AF00;
  background-position: right center;
  color: #fff;
  border-color: #205b00;
}

.btn-secondary-hover {
  background-color: #F1F1F2;
  background-position: right center;
  color: #333;
  border-color: #777;
}
```



图14-10　次要按钮的悬停状态

使用组合图像（image sprites）

为了减少服务器请求的数量，把主要和次要按钮的默认及悬停背景图共同放进一张组合图像里。组合图像是一个单独的图像文件，内含多张背景图。可以使用CSS的background-position属性来确定它的位置，以显示出合适的背景图。

按钮的默认状态在左边，悬浮状态在右边，因此可以在样式表里简单地把每一种状态的背景位置从左换到右。这张组合图像有1000像素宽（一半是默认状态，一半是悬停状态），因此按钮最多可以有500像素宽，如图14-11所示。



图14–11　主要和次要按钮的背景组合图像

最后，我们会给legend和段落添加字体大小的样式规则，为它们创建出合适的视觉比重：

```
legend { font-size:1.5em; }
fieldset p { font-size:1.3em; }
```

应用这些样式后，增强标记看上去就和我们的目标设计一模一样了，如图14-12所示。



图14-12　最终的增强样式

14.2.3　悬停状态增强脚本

基于input的按钮几乎不需要编写脚本：当用户把光标放在某个按钮上时，我们会用JavaScript给按钮应用hover类。这种方法确保了任何通过能力测试的浏览器都能显示出正确的悬停状态反馈。

首先，创建两个变量来捕捉input元素，它们分别对应网页上所有的主要和次要按钮：

```
//找到所有带有主要和次要按钮类的input元素
var primaryButtons = $('input.btn-primary');
var secondaryButtons = $('input.btn-secondary');
```

然后添加hover类，方法是绑定一个函数到input的mouseover事件，并用mouseout事件来移除这个类。为方便起见，jQuery提供了一个hover方法，可同时传递用于mouseover和mouseout事件的函数。下面的例子展示了如何给主要按钮应用这种悬停状态切换方法：

```
//应用hover方法
primaryButtons.hover(
  //第一个函数参数会应用到mouseover事件
  function(){
      $(this).addClass('btn-primary-hover');
  },
  //第二个函数参数会应用到mouseout事件
  function(){
      $(this).removeClass('btn-primary-hover');
  },
);
```

对于次要按钮，构建一个类似的hover方法作用于secondaryButtons变量，并用这个方法切换btn-secondary-hover类的开启和关闭。

现在，input按钮支持了开头的目标设计中指定的行为和视觉外观。接下来，分析如何将带有更复杂视觉格式的按钮从input转变成增强体验里的button。

14.3　创建带有复杂视觉格式的按钮

我们的第二个目标设计建立在第一个例子的基础之上，并带有两个额外的特性："添加"和"忽略"（Ignore）操作按钮的图标，以及用字体粗细的差异来强调主要按钮里的部分文字，如图14-13所示。



图14-13　带有复杂按钮格式的目标设计

在这个案例里，无法让input元素的样式匹配目标设计，因此只会在基础标记里使用input元素，因为我们知道它能可靠地在所有浏览器和设备上工作。在增强体验里，则会运行一个脚本，通过将input替换成一个带有样式的button元素，提供更为丰富的体验。

用button替换input元素时，把input元素的所有属性和JavaScript事件复制到button上，以确保新老元素一一对应。新的button会完全取代input元素的所有功能，因此当脚本运行完毕后，会从页面里彻底移除input。

14.3.1 基础标记和样式

像第一个例子一样，从input元素起步并指派主要和次要按钮类，不过我们会额外给每个input添加一个类来标明各个按钮的用途（action-add和action-ignore）。这些类会传递给增强体验里的button，用来给按钮应用特定的图标：

```
<input type="submit" name="add" value="Add John as a friend" class="btn-
primary action-add" />
<input type="submit" name="ignore" value="Ignore" class="btn-secondary
action-ignore" />
```

为了标明某些按钮文字应该带有粗体样式，使用一个HTML字符来作为定界符。脚本会寻找这个定界符，然后给它前面的文字应用粗体样式。好的定界符通常是一些标点符号或者特殊字符，它们能合情合理地显示在基本体验里，而且足够独特，能被脚本探测到。对这个按钮来说，没有哪种可见字符能朗读正确，所以定界符的最佳选择是一个不间断的空格（ ），它在基本体验里不可见：

```
<input type="submit" name="add" value="Add John as a friend"
class="btn-primary action-add" />

<input type="submit" name="ignore" value="Ignore" class="btn-secondary
action-ignore" />
```

我们为第一个例子创建的基本样式在这里也适用，只有一处需要稍作修改：把button添加到font-family选择器里，让它能继承正确的字体样式：

```
body, input, button { font-family: "Segoe UI", Arial, sans-serif; }
```

基础标记和基本样式到位后，下面来处理浏览器通过能力测试时应用到网页上的增强标记和样式。

14.3.2 增强标记和样式

当浏览器通过能力测试时，用JavaScript把各个input替换成拥有相同属性和JavaScript事件的button元素：

```
<button type="submit" name="add" value="Add John as a Friend" class=
"btn-primary action-add">Add John as a Friend</button>

<button type="submit" name="ignore" value="Ignore" class="btn-secondary
action-ignore">Ignore</button>
```

增强脚本会移除之前插入到基础标记里的定界符（即一个不间断的空格： ），然后把它前面的文字用一个strong元素围起来：

```
<button type="submit" name="add" value="Add John as a Friend" class="btn-
primary action-add"><strong>Add John</strong> as a Friend</button>
```

为了应用图标，脚本会在标记里插入一个class为icon的span元素，位置是在button文字的前面：

```
<button type="submit" name="add" value="Add John as a Friend" class=
"btn-primary action-add"><span class="icon"></span><strong>Add John
</strong> as a Friend</button>
```

和在基本CSS上的做法一样，把之前构建用来给input添加样式的增强CSS作为基础，为增强的button元素指定所需样式。

在目标设计里，主要按钮上的一部分文字是粗体，其余部分则是正常的字形。btn-primary类给所有的主要按钮应用了粗体font-weight属性，所以需要让这个特殊的主要按钮成为一个例外。为action-add类编写一条新的样式规则，专门面向目标设计里的主要按钮，这样网页里的其他主要按钮就仍然是粗体的：

```
button.action-add { font-weight: normal; }
```

这个“input转button”的脚本会用strong标签包围强调的文字：*添加约翰*。默认情况下，大多数浏览器会给strong元素应用一种加粗的字体样式，不过为了确保格式应用正确，用一条简单的规则巩固这个样式：

```
button strong { font-weight: bold; }
```

我们还会给图标的span添加样式，对它们应用inline-block属性和尺寸：

```
button span.icon {
  display: inline-block;
  height: 12px;
  margin-right: 5px;
  width: 16px;
}
```

最后，为每个图标span都设置一张背景图像，具体根据它们父元素的类而定：

```
.action-add span.icon {
  background: url(../images/icon-add.png) 0 0 no-repeat;
}
.action-ignore span.icon {
   background: url(../images/icon-ignore.png) 0 0 no-repeat;
}
```

我们的CSS现在已经支持了复杂按钮格式，接下来编写脚本，实现input到button的转换。

14.3.3　input转button增强脚本

我们的目标是在浏览器通过能力测试时替换基础标记里的特定input元素，用自定义样式的button元素代替。首先，创建一个buttoninputs变量来存放一个引用数组，里面引用的是网页里所有类似button的input元素：

```
var buttoninputs = $('input.btn-primary, input.btn-secondary');
```

用jQuery的each方法遍历数组里的各项，把它们替换成一个新的button元素，还包括这些

input元素的所有属性和绑定事件。each方法接受一个函数作为参数，用于对buttoninputs数组里的各项执行：

```
buttoninputs.each(function(){
 ...
});
```

在这个函数里，我们会编写逻辑代码来获取input的所有属性和事件，把它们指派给新的button元素，然后替换input。

首先，创建一个变量来充当button标记的模板角色，并将input元素的值用于button元素的文本标签：

```
buttoninputs.each(function(){
  var button = $('<button>'+ $(this).val() +'</button>');
});
```

> **注意** 在传递给each方法的函数里，this关键词指的是当前的input元素。通过把this放在一对括号里并添加一个美元符号前缀，把它转变成了一个jQuery对象引用，这样就可以对它应用jQuery方法来操作该元素。举个例子，$(this).val();这条语句会返回input的值。

接下来，获取input元素的所有属性名/值对（技术上来说是nodeName和nodeValue），然后用setAttribute方法把它们应用到新的button上：

```
buttoninputs.each(function(){
  var button = $('<button>'+ $(this).val() +'</button>');
  $.each(this.attributes, function(){
      button[0].setAttribute(this.nodeName, this.nodeValue);
  });
});
```

> **注意** 写作本书时，用jQuery的attr方法在Internet Explorer 8里设置type属性时会产生一个错误。为了避免与Internet Explorer冲突，我们使用了JavaScript原生的setAttribute方法，而不是jQuery的attr快捷方法。就像所有的JavaScript原生属性一样，setAttribute必须使用数组标号来引用button变量，即button[0]。

现在我们已经把input元素的属性重新指派给了button，下面对input元素绑定的所有JavaScript事件执行同样的操作。用jQuery绑定的事件通过data方法保存在内存中，这个方法既能获取数据，也能设置数据。向data方法传递一个参数（在这个案例是events）来获取绑定在input上的所有事件。又因为每个事件（比如onload和onclick）都可以关联多个绑定（或者说函数），所以我们还会遍历这些绑定来确保一个不漏：

```
buttoninputs.each(function(){
  var button = $('<button>'+ $(this).val() +'</button>');
```

```
    $.each(this.attributes, function(){
        button[0].setAttribute(this.nodeName, this.nodeValue);
    });
    if ($(this).data('events')) {
      $.each($(this).data('events'),
          function(eventname, bindings){
              $.each(bindings, function(){
                  button.bind(eventname, this);
              });
           });
     };
  });
```

此时，用原来的input元素属性和事件创建了自定义按钮，现在可以安全地把input元素从标记里移除了。jQuery的replaceWith方法提供了一种干净的方式，能够同时移除input元素和插入新的button元素：

```
buttoninputs.each(function(){
  var button = $('<button>'+ $(this).val() +'</button>');
  $.each(this.attributes, function(){
      button[0].setAttribute(this.nodeName, this.nodeValue);
  });
  if ($(this).data('events')) {
      $.each($(this).data('events'),
          function(eventname, bindings){
              $.each(bindings, function(){
                  button.bind(eventname, this);
              });
          });
  };
  $(this).replaceWith(button);
});
```

现在已经把input转换成button元素了。接下来，添加文本格式（strong标签）和图标的span。重置action-add按钮的内部HTML，方法是在不间断空格字符的位置拆分文本，然后用一个strong元素围住前一部分：

```
//获得对添加按钮的引用
var addButton = $('button.action-add');

//设置添加按钮的HTML
addButton.html(
'<strong>'+addButton.html().split(' ')[0]+'</strong>'+
  addButton.html().split(' ')[1]
);
```

最后，用jQuery的prepend方法给所有按钮添加图标span元素，这个方法会在元素的开头插入HTML：

```
$('button').prepend('<span class="icon"></span>');
```

14.4 使用 input 转 button 脚本

本书附带的代码（位于www.filamentgroup.com/dwpe）包含一个jQuery脚本，它把本章展示的原则（特别是第二个例子里的）包装成一个可以重用的脚本，方便你在自己的项目里使用。要使用这个脚本，只需加入按钮演示页里引用的所有文件，然后调用inputToButton方法即可。调用的对象可以是任何你想转换成button的input元素。

```
$('input#submitForm').inputToButton();
```

inputToButton插件会自动转移类和事件，并给按钮应用悬停类。可以安全地在一切input元素上调用inputToButton方法，因为这个脚本能确保不转换任何非按钮的input类型，比如text、radio或checkbox。

```
$('input').inputToButton();
```

要添加图标span和用不间断空格定界的文字加粗（就像本章对“添加”按钮所做的这样），可以使用额外的脚本逻辑来寻找不间断空格并插入图标span：

```
$('input').inputToButton();

//在 字符处拆分文本，同时删掉它
var splitText = addButton.html().split(' ');

//找到带有action-add类的button，把它的内容替换为如下标记：
//用一个strong元素围住splitText的第一部分，后面跟着splitText第二部分的文字
$('button.action-add')
    .html('<strong>'+ splitText[0]+'</strong> '+ splitText[1]);

//给所有按钮添加图标span，放在文字的前面
$('button').prepend('<span class="icon"></span>');
```

14.5 让按钮更进一步

除了提交数据之外，按钮还经常被应用程序用来切换设置或首选项。切换按钮通常会有一个额外的“按下”状态，让它看上去已被选中。

切换按钮既可以单独作为一个开启/关闭开关使用，也可以组成一组，每次只允许选中一项。如果你认为这听上去像是另一种HTML元素，那你就猜对了：切换按钮的行为与复选框和单选按钮一模一样。

要详细了解如何像切换按钮那样给复选框和单选按钮添加样式，请参阅第15章。本书附带的代码还包括一张演示网页，里面的切换按钮类似于本章展示的这些，参见所附代码checkbox-radiobutton文件夹里的toggle-buttons.html。

按钮的使用和样式

下面这些文章很有帮助，指导我们自定义按钮的工作：

http://t.cn/hCPmr

http://jehiah.cz/archive/button-width-in-ie

http://t.cn/zRlYk8P

http://t.cn/zRlH7v6

http://t.cn/zRlHZYU

自定义样式的按钮能传达增强的视觉复杂性并改善网站的可用性。创建你自己的按钮时，请记住以下这些规则，以确保为所有人提供可用的体验。

- 一开始要创建一个普遍可用的`input`元素，并使用尽可能少的安全样式，以确保所有用户都能访问到关键功能。
- 如果按钮的设计比较简单（没有文本格式或图标），就应该把`input`元素同时用于基本体验和增强体验，并在能力测试通过时应用安全样式。
- 对于那些较为复杂的设计，应该将各个`input`元素转换成`button`，为有能力的浏览器和设备添加样式和视觉反馈。
- 用em设置按钮字体大小和内边距样式，而不是像素，这样可以保留文本大小缩放的能力。
- 为了尽量减少服务器请求，应该为按钮背景使用组合图像。要确保组合图像足够宽，能容纳字体尺寸放大时可能出现的最长的按钮。

第 15 章 复选框、单选按钮和星级评分

多亏了Web标准和高级CSS技巧的广泛使用，现代Web设计里的大多数界面组件现在都具备了自定义的外观和感觉，它们丰富的颜色种类和多层次的背景图像紧密集成于网站的整体视觉样式之中。

然而，表单里的复选框和单选按钮却是这个规律顽固的例外。它们已经落伍了，主要原因是只有少数浏览器才内建对它们应用CSS样式的支持。让情况更加复杂的是，各种浏览器在渲染复选框和单选按钮时会在尺寸、间距和外观上略有不同。

对许多人，包括我们的大多数客户来说，原生复选框和单选按钮里固有的可用性功能（包括可访问性、键盘支持、标签交互和普遍的浏览器兼容性）已经足够显著，让自定义样式所带来的好处相形见绌。因此，我们在大多数情况下一般会使用原生的元素，同时接受这个事实：我们无法让它们的外观匹配我们的设计系统。

不过，我们遇到过一些设计项目，它们确实需要一种自定义的设计解决方案，包括改变复选框和单选按钮的颜色、样式和大小。以下是一些例子。

- 触摸屏移动设备或者自助服务终端（kiosk）。我们为它们创建的切换组件具有自定义的样式，尺寸偏大，适合手指操作，但仍然需要像原生控件那样捕捉用户的数据输入。
- 图表应用程序。图表里用颜色编码的复选框不仅扮演着图例的角色，还用于切换图中各项的开启和关闭。
- 带有排序控件的搜索结果页。控件包含了日期、相关性和流行程度等链接，用户选择单个选项来给数据排序（类似于一组单选按钮）。
- 简单的复选框或单选按钮。它们需要自定义的样式来匹配网站的品牌风格和视觉设计，特别是那些使用深色或彩色背景的网站。

当原生元素无法提供某种程度的视觉控制或功能，必须使用自定义设计的表单元素时，就需要制定一个方案，让它具备与被取代元素相同的可访问性和易用性。

这一章会展示如何使用渐进增强的方法，把标准HTML复选框和单选按钮转变成尽可能轻量的自定义设计版本。我们还会逐步介绍构建自定义星级评分组件（star rating）的过程，因为它也用到了许多相同的技巧和脚本。

15.1　X光透视

自定义输入元素的一个常见用途是调查表，表中的问题、分组复选框和单选按钮匹配某种特定视觉设计的观感。我们的目标设计如图15-1所示。

图15-1　应用了自定义复选框和单选按钮样式的调查表设计

用X光透视分析这个案例里的设计非常容易：很明显应该选择使用标准的复选框和单选按钮作为我们的基础标记。凭借它们的自身能力，复选框和单选按钮能捕捉用户输入并显示反馈。我们的目标是利用这种原生的功能，从而不必用JavaScript再创建一遍。

搜索能在不牺牲原生元素丰富功能的前提下实现自定义样式的最佳方法。一番快速的搜索后，几个脚本浮现了出来，它们巧妙地组合使用JavaScript和CSS来创建自定义样式的复选框和单选按钮，方法是隐藏原有的`input`，然后插入那些能够有效添加样式的自定义元素标记进行替换。这些脚本通常相当复杂，因为它们要重新创建原生复选框或单选按钮所具备的全部功能、行为、状态追踪、键盘事件和可访问特性。如果是不理解ARIA的旧款屏幕阅读器，或者浏览器不支持图像，这些技巧就会失效，留下不能用的`input`。虽然替换有时是必须的（第17章介绍自定义下拉列表框时会介绍），但它引入了代码复杂性，而我们想找到一种更为优雅的解决方案。

探寻其他主意时，我们想起了一个关键的特性，它可以解决我们的问题：在所有的主流浏览器里，点击关联某个复选框或单选按钮`input`的`label`时，产生的效果和点击表单元素自身相同。它会切换复选框的选中状态，或者选择单选按钮。

理论上，这种事不应该发生。HTML规范写道：当一个`label`元素获得焦点时，它把焦点传给它所关联的控件。但是，在我们测试的每一种浏览器里，点击标签会把焦点和*点击*事件同时传给关联的复选框或单选按钮。这种浏览器行为给了我们一个至关重要的挂钩，让我们能在很大程

度上借用原生元素，为增强体验制作轻量但有着自定义视觉效果的复选框和单选按钮。

我们知道能单纯依靠label元素来控制原生复选框或单选按钮的状态，而不必要求用户直接点击实际的input元素。可以在增强体验里利用label的这个特性给它应用样式，使它看起来像一个自定义的input，并能提供视觉反馈。

做一些CSS定位工作后，就可以把label元素堆叠到原生复选框或者单选按钮的上方了，如图15-2所示。通过给label添加一张带有自定义复选框或单选按钮设计的背景图像，实际上就遮蔽了原生元素，尽管它仍然是可见的。做这一切的必要条件是input和标签在基础标记里有着正确的语义配对：换句话说，标签的for属性值要匹配input的id值。



图15-2　标签和它的自定义背景图像被CSS放置在原生复选框的上方

通过这种方法，我们让原生复选框保持可见，并处在它正常的网页位置上，而不是用CSS把它定位到屏幕之外或者隐藏起来。图像可用时，背景图像就会显示在标签上，能看到自定义样式，同时遮住了原生的input。浏览器不支持图像时，只要标签的背景色被设置成透明，原生input就能透过标签看到，这样的话控件仍然可用，如图15-3所示。

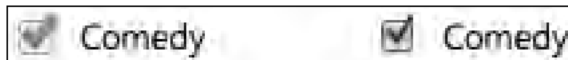


图15-3　增强版的自定义复选框，分别是浏览器启用图像（左图）和禁用图像（右图）时

这个阶段，我们已经有了一套清晰的策略来增强一组HTML复选框和单选按钮。对我们网站的每位访问者（即使用的是最古老的浏览器，最精简的可上网设备，或者系统选择性禁用了某些功能），我们都会输出简单的语义化HTML，并且有信心让所有人都能访问它，如图15-4所示。

图15-4　基本体验里所见的复选框和单选按钮

使用标签创建增强的自定义样式复选框或单选按钮能够保留原生input的所有功能、固有的键盘支持，以及可访问特性。这样做就无需再使用ARIA属性了，因为所有用户（包括屏幕阅读

器用户）仍然是在和原生的表单元素交互。使用这种方法后，增强体验里几乎完全不需要再添加额外的标记，脚本也能写得简单和轻量。

15.2　创建自定义复选框

确立目标设计和最终目的后，接下来分析如何构建基础标记，以及如何用CSS和一小段脚本把标记增强成自定义的表单控件。

15.2.1　基础标记

首先为调查问题的复选框部分创建基本HTML标记。

用一个form标签围住调查问卷，这样就能提交它了。在它的内部，把复选框控件纳入一个fieldset进行归组，并将legend标签用于问题的文字部分。

关联各个input和label至关重要，方法是设置标签的for属性，使它匹配复选框input的id。这些元素必须正确配对才能通过点击标签来切换复选框的状态：

```
<form action="moviepoll.php" action="post">
  <fieldset>
      <legend>Which genres do you like?</legend>

      <input type="checkbox" name="genre" id="action" value="action" />
      <label for="action">Action / Adventure</label>

      <input type="checkbox" name="genre" id="comedy" value="comedy" />
      <label for="comedy">Comedy</label>

      <input type="checkbox" name="genre" id="epic" value="epic" />
      <label for="epic">Epic / Historical</label>

      <input type="checkbox" name="genre" id="science" value="science" />
      <label for="science">Science Fiction</label>

      <input type="checkbox" name="genre" id="romance" value="romance" />
      <label for="romance">Romance</label>

      <input type="checkbox" name="genre" id="western" value="western" />
      <label for="western">Western</label>
  </fieldset>
</form>
```

注意　在屏幕阅读器里，fieldset的legend文字会在里面的各个input之前被重复朗读，所以最好让它保持简洁，同时又有描述性。

在完全不添加CSS样式的情况下，这些复选框和标签对很清楚地与问题归组在一起，使调查表对所有人都可用，如图15-5所示。

Which genres do you like?
☑ Action / Adventure ☑ Comedy ☐ Epic / Historical ☐ Science Fiction ☐ Romance ☐ Western

图15-5 无样式的复选框基础标记

为了让它看起来与我们的网站更加一致并添加少许视觉样式，因此应用一些非常简单的CSS规则，它们对几乎所有浏览器都是“安全”的。

- 给body加上一列字体，使它匹配我们的网站设计。
- 将legend加粗，使问题和选项之间有更好的视觉层次。
- 给legend四周和每个标签的右侧都加上一点外边距，从而在视觉上更好地分隔这些选项，并增加各个复选框和其标签之间的视觉联系，使网页更容易阅读。
- 我们的“安全”样式在基本样式表里看起来如下所示：

```
body { font-family: "Segoe UI", Arial, sans-serif; }

fieldset { margin-top: 1em; margin-bottom: 1em; border:1px solid #ddd; }
legend { font-weight:bold; }
label { margin-right:1.2em; }
```

- 在网页上看起来则如图15-6所示。

Which genres do you like?
☑ Action / Adventure ☑ Comedy ☐ Epic / Historical ☐ Science Fiction ☐ Romance ☐ Western

图15-6 应用安全样式后的基础复选框

注意 我们没有在基本的安全样式上花太多心思，因为更好的做法是使CSS保持简单，尽可能让各种设备使用原生的方式渲染。你会看到我们不指定这里任何元素的字体大小或者尺寸，因为想让浏览器自己判断如何才能最好地呈现这段HTML。另外，我们会偶尔但有目的地使用外边距，因为相信它大大增加了网页的可用性，但是我们也明白，许多老式或移动浏览器可能会忽略外边距的样式规则。

15.2.2 增强标记和样式

建立基础标记后，现在可以确定浏览器通过能力测试后增强脚本应该叠加上的标记变化和高级样式了。

这个自定义复选框十分简单，只需要做一次标记增强：增强脚本会添加一个带有custom-checkbox类的包装器div：

```
<div class="custom-checkbox">
  <input type="checkbox" name="genre" id="check-1" value="action" />
```

```
  <label for="check-1">Action / Adventure</label>
</div>
```

这个类会用来应用我们的增强CSS样式。首先，div会被相对定位，这样就能把label和复选框绝对定位在这个容器内：

```
.custom-checkbox { position:relative; }
```

接下来，绝对定位input，并将它从容器的左上角向内移动几像素，帮助它与标签文本对齐。然后给标签设置z-index来确保它位于上层：

```
.custom-checkbox input {
   position:absolute;
   left:2px;
   top:2px;
   margin:0;
}
```

接下来，为label编写CSS规则，将它显示成一个位于input（z-index为1）前面的块级元素。给标签增加30像素的左侧内边距，给处于文本标签左侧的自定义复选框背景图像留出空间。再添加一条规则来确保整个标签在所有浏览器里看上去都是可点击的，具体做法是把cursor设为pointer：

```
.custom-checkbox label {
  display:block;
  position:relative;
  font-size:1.3em;
  padding-right:1em;
  line-height:1;
  padding:.5em 0 .5em 30px;
  margin:0 0 .3em;
  cursor:pointer;
  z-index:10;
}
```

然后添加一张组合图像，这张自定义设计的背景图像代表了标签的各种状态。

提示　用单张图像代替多张能降低服务器请求数量，而且会预先缓存各种状态，从而使网页能够更快地加载。

使用这种方法时有两个必要条件需要牢记，它们确保了组合图像能完全遮盖原生的复选框。

- 组合图像必须有不透明的背景，这样它们就能完全遮住原生的input。
- 组合图像里的每一张子图像都必须足够大，能完全遮住任何交互状态下的复选框。举个例子，当你在苹果Mac电脑上使用火狐浏览器时，复选框在获得焦点后会“发光”，显示出3像素的蓝色边框。组合图像里的子图像必须足够大，能遮住这种效果。

我们的组合图像垂直堆叠了4种不同的复选框状态，自上而下分别是未选中、未选中时悬停、已选中和已选中时悬停。可以按照你认为合适的间距分隔这些组合图像，但它们之间的距离应该

足够远，以防止别的状态显露出来，因为标签的高度会随着字体增大或自动换行而增加。考虑到这个例子的目的，我们给各张状态图像均匀地设置了100像素的间距，如图15-7所示。

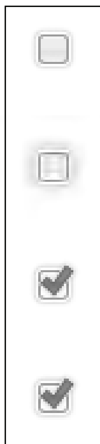


图15-7 复选框的组合图像对所有交互状态都做了视觉处理，把它们堆叠起来，并在各个状态之间留出了很大的空间

我们设置了一条通用规则，把复选框组合图像指派给了label：

```
.custom-checkbox label {
  background:url(../images/checkbox.gif) no-repeat;
}
```

现在，可以加入CSS规则来移动复选框组合图像的背景位置，以用于未选中、未选中时悬停、已选中和已选中时悬停这些状态。因为每种状态都在组合图像里被均匀地分隔开，所以只需要每次调整100像素的vertical position就可以了。理想情况下，使用CSS的:hover伪类来指派悬停状态，但不是所有的现代浏览器都支持标签元素的:hover，因此会给mouseover添加一个名为hover的类，然后在mouseout上移除它。

```
.custom-checkbox label {
  background-position:-10px -14px;
}
.custom-checkbox label.hover, .custom-checkbox label.focus {
  background-position:-10px -114px;
}
.custom-checkbox label.checked {
  background-position:-10px -214px;
}
.custom-checkbox label.checkedHover, .custom-checkbox label.checkedFocus {
  background-position:-10px -314px;
}
```

此时，所有4种复选框状态看上去都和原设计一样了，如图15-8所示。



图15-8　增强的复选框设计，通过改变组合图像的背景位置创建各种状态

虽然悬停状态的视觉转换很有帮助，但不是所有用户都能辨别它们。举个例子，为了让获得焦点时的状态对用键盘导航的用户更明显，给当前获得焦点的选项添加一圈浅灰色的点状轮廓线（如图15-9所示），方法是在标签上添加一个focus类：

```
.custom-checkbox label.focus { outline: 1px dotted #ccc; }
```



图15-9　围绕着标签的点状轮廓线有助于键盘导航

制作出各种交互状态的标记和样式后，我们接下来会编写脚本，在用户悬停鼠标和点击自定义复选框时切换标签的交互状态。

15.2.3　复选框脚本

用来增强原生复选框的脚本非常简单。首先，收集所有的复选框，方法是寻找任何type为checkbox的input：

```
var checkboxes = $('input[type=checkbox]');
```

然后使用jQuery的each方法遍历各个复选框寻找对应的label（条件是它的for属性匹配该复选框的id），然后把两者放入一个带有custom-checkbox类的div里。同时趁着手头上有label，还会给它指派mouseover和mouseout事件，作用是当用户把鼠标悬停在它上方时切换hover类：

```
//遍历各个复选框input
checkboxes.each(function(){
  //找到关联的标签元素
  var label = $('label[for='+ $(this).attr('id) +']');
  //给标签绑定一个mouseover事件
  label.mouseover(function(){
     //添加hover类
     $(this).addClass('hover');
     //检查复选框是否有.checked类
     if( $(this).is('.checked') ){
         //添加checkedHover类
```

```
            $(this).addClass('checkedHover');
        }
        });
    //给标签绑定mouseout事件
    label.mouseout(function(){
        //移除所有的hover类
        $(this).removeClass('hover checkedHover');
    });
    //把复选框和标签放入一个自定义复选框div里
    $(this).add(label).wrap('<div class="custom-checkbox"></div>');
});
```

接下来，创建一个自定义的updateState事件，检查input当前是否已被选中。如果已选中，脚本就会添加checked类,将组合图像定位到正确的状态。如果未选中,就移除任何现有的checked、checkedHover或checkedFocus类，使自定义复选框看起来是未选中的：

```
//给所有复选框绑定自定义的updateState事件
checkboxes.bind('updateState', function(){
  //找到关联的标签元素
  var label = $('label[for='+ $(this).attr('id') +']');
  //检查复选框是否已被选中
  if ( $(this).is(':checked') ) {
      //给标签元素添加checked类
      label.addClass('checked');
  }
  //如果复选框未被选中
  else {
      //从标签上移除一些类
      label.removeClass('checked checkedHover checkedFocus');
  }
});
```

绑定这个自定义事件本身实际上不会做任何事，因为它不是click或mouseover这样的原生事件。为了让updateState能够运行，我们必须自己触发它。脚本首次运行时，使用jQuery的trigger方法来运行updateState函数，设置所有自定义复选框以匹配原生复选框的选中状态：

```
//在网页载入时立即触发updateState
checkboxes.trigger('updateState');
```

然后,给每个复选框都绑定一个click事件,只要点击复选框input或label就运行updateState函数。使用jQuery的简化方法click来绑定点击事件，它的作用和bind('click')是一样的：


```
//给复选框绑定一个click事件
checkboxes.click(function(){
   //点击复选框时触发updateState
   $(this).trigger('updateState');
});
```

为了给带焦点状态添加点状轮廓线，在某个input获得焦点时指派focus类，用的同样是该事件的简化方法：

```
//给复选框绑定一个focus事件
checkboxes.focus(function(){
```

```
  //找到关联的标签元素
  var label = $('label[for='+ $(this).attr('id') +']');
  ///给标签添加focus类
  label.addClass('focus');
  //如果这个input是选中的
  if( $(this).is(':checked') ){
     //添加checkedFocus类
     $(this).addClass('checkedFocus');
  }
});
```

然后在input失去焦点时移除focus类：

```
//给复选框绑定一个blur事件
checkboxes.blur(function(){
  //找到关联的标签元素
  var label = $('label[for='+ $(this).attr('id') +']');
  //从标签上移除focus相关的类
label.removeClass('focus checkedFocus');
});
```

这些从基本到增强的标记、样式和脚本就是用于调查表复选框的完整代码。应用这些增强样式后的最终设计如图15-10所示。

Which genres do you like?

- [x] Action / Adventure
- [x] Comedy
- [ ] Epic / Historical
- [ ] Science Fiction
- [ ] Romance
- [ ] Western

图15-10　带有自定义复选框的最终调查表设计

15.3　创建自定义单选按钮

我们会给单选按钮使用和复选框差不多的HTML标记，并遵循类似的操作方法，大量重复使用样式和脚本来增强它们。

15.3.1　基础标记

和复选框一样，我们会给问题创建一个带legend的fieldset，然后将一组单选按钮input和label对用于各个选项：

```
<fieldset>
  <legend>Caddyshack is the greatest movie of all time, right?</legend>

  <input type="radio" name="opinions" id="totally" value="totally" />
  <label for="totally">Totally</label>

  <input type="radio" name="opinions" id="no-way" value="no-way" />
  <label for="no-way">You must be kidding</label>

  <input type="radio" name="opinions" id="whats-caddyshack"
  value="whats-caddyshack" />
 <label for="whats-caddyshack">What's Caddyshack?</label>

</fieldset>
```

在完全不带CSS的情况下，这些单选按钮控件很清楚地与它们对应的问题归组在一起，而且完全可用。如果我们把这些单选按钮添加到之前创建的表单里，那么fieldset、legend和label元素就会继承为复选框创建的那些“安全”样式，如图15-11所示。

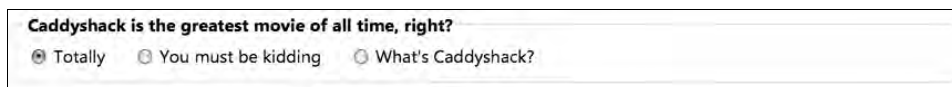


图15-11 应用安全样式后的单选按钮基础标记

15.3.2 增强标记和样式

和复选框一样，增强脚本会用一个带有custom-radio类的div来围住每一对单选按钮input和label，这样就可以添加增强样式和行为了：

```
<div class="custom-radio">
  <input type="radio" name="opinions" id="totally" value="totally" />
  <label for="totally">Totally</label>
</div>
```

然后为单选按钮创建一张背景组合图像（同样要确保组合图像有不透明的背景，每种状态要大到能遮住原生的浏览器反馈，状态图像要垂直排列，并用标准的100 px增量分隔开），如图15-12所示。

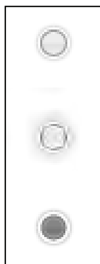


图15-12 包含未选中、悬停和已选中状态的自定义单选按钮组合图像

你可能会注意到一个不同之处，那就是单选按钮没有之前为复选框创建的已选中时悬停这种状态（就像.custom-checkbox label.checkedHover）。单选按钮在选中时不允许用户把它切换到未选中状态，所以这一块的视觉反馈是不必要的。

因为自定义单选按钮的图像和复选框同样位于原生input的上方，所以可以把单选按钮规则添加到增强样式表里的复选框规则上。

我们会添加custom-radio类选择器到封装容器、input和标签的规则上，而不需要修改这些CSS规则：

```
.custom-checkbox, .custom-radio {
  position:relative;
}
.custom-checkbox input, .custom-radio input {
  position:absolute;
  left:2px;
  top:2px;
  margin:0;
}
.custom-checkbox label, .custom-radio label{
  display:block;
  position:relative;
  font-size:1.3em;
  padding-right:1em;
  line-height:1;
  padding:.5em 0 .5em 30px;
  margin:0 0 .3em;
  cursor:pointer;
}
```

另外，为custom-radio类设置背景组合图像的路径：

```
.custom-radio label {
  background: url(../images/radio button.gif) no-repeat;
}
```

我们特地把单选按钮子图像的间距设置得和复选框组合图像一致，这样它们的各种状态就能共用同一条CSS图像定位规则。我们只需要给各条规则添加custom-radio标签的选择器即可。因为checkedHover状态只适用于复选框，所以我们会让这条规则保持不变：

```
.custom-checkbox label,
.custom-radio label {
  background-position:-10px -14px;
}
.custom-checkbox label.hover,
.custom-checkbox label.focus,
.custom-radio label.hover,
.custom-radio label.focus {
  background-position:-10px -114px;
}
.custom-checkbox label.checked,
.custom-radio label.checked {
  background-position:-10px -214px;
```

```
}
.custom-checkbox label.checkedHover, .custom-checkbox label.checkedFocus {
  background-position:-10px -314px;
}
```

此时，为全部三种单选按钮状态（未选中、未选中时悬停和已选中）提供所需样式的代码就完成了，如图15-13所示。

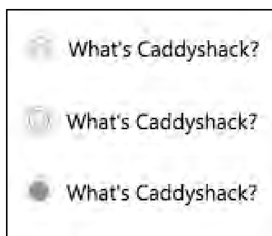


图15-13 创建全部三种单选按钮状态的方法是改变组合图像的背景位置

15.3.3 单选按钮脚本

我们的自定义复选框和单选按钮既有着相似的设计，又有相似的行为，所以合理的做法是把单选按钮的功能逻辑整合到之前为复选框创建的脚本之中。

首先，修改一开始的选择范围，把单选按钮也包括进来，用checksRadios的变量名代替checkboxes：

```
//找到页面上所有的复选框和单选按钮
var checksRadios = $('input[type=checkbox],input[type=radio]');
```

然后修改第一个循环，用一个带有恰当类名的div围住我们的input/label对：

```
//遍历各个input
checksRadios.each(function(){
  //找到关联的标签元素
  var label = $('label[for='+ $(this).attr('id') +']');
  //给标签绑定一个mouseover事件
  label.mouseover(function(){
      //添加hover类
      $(this).addClass('hover');
      //检查复选框/单选按钮是否有.checked类
      if( $(this).is('.checked') ){
         //添加checkedHover类
         $(this).addClass('checkedHover');
     }
 });
  //给标签绑定mouseout事件
  label.mouseout(function(){
     //移除所有的hover类
     $(this).removeClass('hover checkedHover');
 });
  //获取input的类型，用作类名后缀
```

```
    var inType = input.attr('type');
    //用带有恰当类的div围住input和label
    $(this).add(label).wrap('<div class="custom-'+ inType +'"></div>');
  });
```

我们的updateState事件需要引用新的checksRadios变量来涵盖复选框和单选按钮。updateState事件所包含的脚本代码不需要做任何修改：

```
//给所有的复选框和单选按钮绑定自定义的updateState事件
checksRadios.bind('updateState', function(){
 ...
});
```

类似地，在网页加载时触发checksRadios上的updateState事件：

```
//网页加载时触发所有复选框和单选按钮上的updateState
checksRadios.trigger('updateState');
```

click事件需要做一点小小的更新，以确保点击某个单选按钮时，其他同属一组的单选按钮也会更新，显示出未选中的状态。因为单选按钮原生就能控制其他按钮的选中状态，所以只需触发updateState事件，触发对象是和被点击单选按钮拥有相同name属性的所有input（当然，也包括被点击的单选按钮）：

```
//给复选框和单选按钮绑定一个click事件
checksRadios.click(function(){
  //获取此input的name属性
  var inName = $(this).attr('name');
  //触发updateState，范围是name相同的所有input
  $('input[name='+ inName +']').trigger('updateState');
});
```

最后，focus和blur事件无需修改就能支持单选按钮，我们会简单地通过给checksRadios变量绑定事件来添加它们：

```
//给复选框和单选按钮绑定一个focus事件
checksRadios.focus(function(){
 ...
});
//给复选框和单选按钮绑定一个blur事件
checksRadios.blur(function(){
...
});
```

现在我们的脚本同时支持了复选框和单选按钮input，并且提供了一种可复用的干净方式，在那些通过能力测试的浏览器里把基本体验增强成一组格式丰富的自定义控件，如图15-14所示。

这是一种非常健壮的方法，它能够优雅地缩放，即使用户选择把他们的字体大小设置得比默认尺寸大上几级也是如此。（使用高级CSS样式构建组件时，测试这一点很重要。）

最棒的是，如果访问者通过了能力测试，但是出于某种原因看不到浏览器显示的图像，那么这种技巧能够安全地转至备用的原生input，而它们是完全可见和可访问的，如图15-15所示。

图15-14　增强自定义复选框和单选按钮

图15-15　禁用浏览器图像之后的增强自定义复选框和单选按钮

15.4　让自定义 input 更进一步：星级评分组件

用定制复选框和单选按钮的技巧创建了一个星级评分组件。评分组件基于预先定义好的尺度，收集和显示用户的主观反馈。其中，星级评分组件尤其流行，它的尺度一般由5颗星构成，

显示出用户有多么喜欢某种产品或服务（比如5颗星=很喜欢，1颗星=很不喜欢，有时还会有一个“忽略”选项）。数以百万计的用户使用星级评分组件进行在线投票，评价亚马逊网站上的商品，或者给Netflix里的电影打分，如图15-16所示。



图15-16　星级评分组件的设计

为了和电影这个主题保持一致，我们会创建一个组件，用户可以给他们看过的电影评分，范围从“不值一看”（unwatchable）到“奥斯卡级”（Oscar-worthy）。

因为用户被限制只能选择一组选项中的一个，所以基础标记的理想选择是一组单选按钮。但是，评分组件自身有一些不寻常的重要特点，需要考虑。

- 评分组件通常被视为独立的组件，在用户选择某一分值后用JavaScript将数据异步提交回服务器。为了确保评分功能在没有JavaScript时也能工作，需要在基础标记里添加一个提交`input`，然后调整脚本，把它从增强体验里移除。
- 和更简单的标准单选按钮或复选框不同，评分组件的尺度是累加式的（就是说，当用户把光标悬停在第4颗星上时，第1、2和3颗星也处于“开启”状态）。我们需要基于自定义复选框和单选按钮的脚本来编写一个新脚本，它将包括一种能提供恰当反馈的机制。

15.4.1　基础标记

评分组件单选按钮组的基础标记和“安全”样式应该完全遵循我们为自定义单选按钮概括的流程。要确保：

- 给所有的单选按钮`input`都指派一个共同的`name`属性和值，使它们成为一个组；
- 给所有的`input`都指派一个唯一的`id`，然后给各个关联的标签指派一个引用此`id`值的`for`属性。

用于增强的标记、样式和脚本也和标准自定义单选按钮元素非常相似。下面关于悬停状态处理的讨论中会详细介绍它们。

为了让用户在缺少JavaScript支持时也能提交评分，在基础代码的每个`fieldset`里都加入一个提交`input`：

```
<fieldset>
  <legend>Star Wars</legend>
  <input type="radio" name="star-wars" id="star-wars-Unwatchable"
  value="Unwatchable" />
  <label for="star-wars-Unwatchable">Unwatchable</label>

  <input type="radio" name="star-wars" id=" star-wars-Bad" value="Bad" />
  <label for="star-wars-Bad">Bad</label>

  <input type="radio" name="star-wars" id="star-wars-OK" value="OK" />
  <label for="star-wars-OK">OK</label>

  <input type="radio" name="star-wars" id="star-wars-Good" value="Good" />
  <label for="star-wars-Good">Good</label>

  <input type="radio" name="star-wars" id="star-wars-Oscar-worthy"
  value="Oscar-worthy" />
  <label for="star-wars-Oscar-worthy">Oscar-worthy</label>

  <input type="submit" id="star-wars" value="Rate this" class= "rating-submit" />
</fieldset>
```

应用评分基础标记后，它在支持基本体验的浏览器里如图15-17所示。

图15-17　电影评分组件的基本体验，内含“给它打分”（Rate this）按钮

因为默认的HTML渲染会把一组单选按钮和标签显示在同一行，所以这个没有启用CSS或JavaScript的基本体验看起来仍然是一个由差到好的水平连续区间，十分接近增强体验。

15.4.2　增强标记和样式

增强版的评分组件只会输出到支持JavaScript的浏览器上，在那里用Ajax提交评分是安全的。启用了Ajax后，提交按钮就不是必需的了，因此在增强样式表里把它们的`display`属性设成`none`来隐藏它们。给每个`input`都指派一个`rating-submit`类，这样就可以用这个类一次性隐藏全部的`input`，方法是给增强样式添加下面这一行：

```
.rating-submit { display: none; }
```

评分组件单选按钮的样式和行为规则与标准的单选按钮稍有不同（举个例子，它们的背景图像是星级图，文本标签则是隐藏的），因此归组input和标签对的包装器div需要有一个独特且有描述性的类。作为评分组件，我们会使用rating-option：

```
<div class="rating-option">
  <input type="radio" name="star-rating-opt" id="OK-1" value="OK" />
  <label for="OK-1">OK</label>
</div>
```

带有rating-option类的input和label元素使用的增强样式和自定义单选按钮样式可以共享定位和块级属性，不过也有一些例外。

首先，要把input上的top位置值从2 px调整为3 px，以确保它能完全被星级图标遮住：

```
.rating-option input {
  position: absolute;
  left: 2px;
  top: 3px;
  margin: 0;
}
```

我们还会调整组件的标签样式来引用星级背景组合图像，隐藏标签文本（方法是把一个负的文本缩进值和overflow: hidden组合使用，使文本对屏幕阅读器保持“可见”），并微调内边距、外边距和尺寸来适应星级图标：

```
.rating-option label {
  background: url(../images/star.gif) no-repeat -14px -11px;
  display: block;
  position: relative;
  width: 25px;
  height: 25px;
  margin: 0;
  cursor: pointer;
  text-indent: -99999px;
  overflow: hidden;
}
```

已选中状态和悬停状态里设置的背景图像位置也应当加以调整，使其适合星级背景组合图像：

```
.rating-option label.checked {
  background-position: -14px -211px;
}
.rating-option label.hover {
  background-position: -14px -111px;
}
```

15.4.3　编写星级评分组件脚本

虽然基于的标记和自定义单选按钮相同，但是这个增强星级评分组件有一些行为上的变化需要在脚本里实现。

首先，用户悬停于或者点击某个星级标签时，它前面的所有同组标签也应该添加样式。另外，因为星级评分组件里的标签文本是隐藏在视野之外的，所以需要给标签添加一条工具提示，让用户能通过悬停了解每个星级所代表的值。最后，用户选中某个星级时，用Ajax自动把评分信息发送给服务器，这样用户无需点击“提交”按钮。

为了支持这些行为变化，我们需要以多种方式改动脚本。相比试着只编写一个脚本，然后在里面加入明确区分两种情况所需的所有条件逻辑，我们的选择是复制原来的自定义复选框和单选按钮脚本，将它修改后用于星级评分组件。

首先，把一开始的元素选择修改成只获取单选按钮，同时将变量名由`checksRadios`改为`starRadios`。在整个脚本里，使用这个`starRadios`变量来指代我们所有的`input`。

```
//找到网页上的所有单选按钮
var starRadios = $('input[type=radio]');
```

需要对一开始的单选按钮循环进行修改。

- 给每个函数都传递一个`index`参数，它提供了一个逐渐增大的数字，每循环一次就会递增`1`。
- 把这个`index`数值存在它的`label`里，方法是使用一个名为`data`的特殊jQuery属性，这样我们就可以设置和获取某个HTML元素的值。
- 悬停时，切换标签自身的`hover`类，以及它前面所有处在同一行的星级评分标签：

```
//循环遍历各个单选按钮，传递index参数
starRadios.each(function(index){
  //找到关联的标签元素
  var label = $('label[for='+ $(this).attr('id') +']');
  //把标签的index数值存放在内存里以备将来使用
  label.data('index', index);
  //给标签绑定一个mouseover事件
  label.mouseover(function(){
      //循环遍历各个同组单选按钮
  starRadios.each(function(){
      //找到该单选按钮的标签，将它保存为oLabel变量
      var oLabel = $('label[for='+$(this).attr('id')+']');
      //如果oLabel就是悬停的按钮，或者在它之前
      if(oLabel.data('index') '<= index) {
          //给它添加hover类
          oLabel.addClass('hover');
      }
   });
 });
 //给标签绑定mouseout事件
 label.mouseout(function(){
      //循环遍历各个同组单选按钮
      starRadios.each(function(){
          //找到该单选按钮的标签，将它保存为oLabel变量
          var oLabel = $('label[for='+$(this).attr('id')+']');
          //无需检查index编号，移除类
          oLabel.removeClass('hover');
      });
 });
```

```
        //用带有rating-option类的div围住input和标签
        $(this).add(label).wrap('<div class="rating-option">'</div>');
    });
```

updateState事件需要引用新的starRadios变量。updateState事件包含的脚本也需要做类似的修改，修改成我们给各个循环创建的变量，从而使每个单选按钮都能影响它前面按钮的外观。这一次，移除同组各个标签上所有hover和checked类，然后给之前的所有同组星级标签添加checked类：

```
//给所有单选按钮都绑定自定义的updateState事件
starRadios.bind('updateState', function(){
  //保存对此标签index值的引用
  var index = $('label[for='+$(this).attr('id')+']').data('index');
  //检查被点击的单选按钮是否已选中
  if ( $(this).is(':checked') ){
      //循环遍历各个单选按钮
      radioSet.each(function(){
          //找到该单选按钮的标签，将它保存为oLabel变量
          var oLabel = $('label[for='+$(this).attr('id')+']');
          //移除hover和checked两个类
          oLabel.removeClass('hover checked');
          //如果oLabel就是悬停的按钮，或者在它之前
          if(oLabel.data('index') <= index) {
              //添加checked类
              oLabel.addClass('checked');
          }
      });
  }
});
```

我们会在网页加载时触发checksRadios的updateState事件，这样所有现存的星级评分值对用户都可见：

```
//在网页加载时触发所有单选按钮的updateState事件
starRadios.trigger('updateState');
```

因为我们的updateState事件能切换与某个单选按钮同组的所有按钮的状态，所以click事件就可以简单地在自己身上触发updateState事件。

不过，确实有必要给click事件添加一些额外的逻辑来向服务器提交星级评分数据。可以用jQuery的ajax函数轻松做到这点：根据标记定义Ajax请求的参数，用表单的method属性指定它是作为get还是post请求发送（在这个案例里应该是post请求，因为数据保存在服务器上），并用它的action属性指定请求的URL。使用jQuery的serialize方法，把表单数据以键值对的形式发送：

```
//给单选按钮绑定一个click事件
starRadios.click(function(){
  //触发这个单选按钮的updateState事件
  $(this).trigger('updateState');
  //获取上级form元素
  var form = $(this).parents('form');
  //用Ajax发送表单数据
  $.ajax({
      type: form.attr('method'),
      url: form.attr('action'),
```

```
      data: form.serialize()
    });
  });
```

现在，点击单选按钮，它们的状态就会保存到服务器上。

focus和blur事件不需要修改就能支持星级评分行为。它们只是简单地切换标签的focus类：

```
//给单选按钮绑定一个focus事件
starRadios.focus(function(){
  label.addClass('focus');
});
//给单选按钮绑定一个blur事件
starRadios.blur(function(){
  label.addClass('focus');
});
```

15.5　使用自定义 input 和星级评分脚本

这本书附带的可下载代码（位于www.filamentgroup.com/dwpe）包含了两个jQuery插件：jQuery.customInput.js和jQuery.starRating.js。它们提供了创建自定义复选框、单选按钮和星级评分组件的简单方法。

要使用jQuery.customInput.js脚本，只需引用自定义input演示页里的那些相关文件，然后在任何你想定制的复选框或单选按钮input上调用customInput方法即可。如果把这个脚本应用到所有的input上，它会表现得足够智能，只增强复选框和单选按钮，而不会去碰文本或者提交input：

```
$('input').customInput();
```

jQuery.starRating.js脚本有着相同的工作方式，不过它只有在单选按钮组上才能生效。

```
$('input').starRating();
```

这两个插件都能接受任意的jQuery选择器，使你能按照你想要的特性划定input的选择范围。举个例子，你只想对某些input调用starRating方法，它们处于一个带有rating类的div内：

```
$('.rating input').starRating();
```

这两个插件的设计原则是尽可能的简单和轻量，因此它们不包含任何的配置选项或方法。你要做的就是在目标input上调用这些方法，然后它们就会被增强成自定义的版本。

本书详细讨论过，我们坚信每次需要扩展某个原生元素时，应该首先挖掘语义化HTML和CSS的全部潜力，然后才寻求从头创建一个新的元素。

在这个案例里，我们给基础HTML标记应用了一些创造性的CSS和一点点脚本代码，创建出自定义样式的复选框和单选按钮表单控件，因而完全不必重新创建这些元素。这么做不仅节省了大量的工作，还保持了原生元素的可访问性、键盘快捷方式和行为，因此我们确信这个技巧能在各种各样的浏览器上可靠地工作。

还有一个附带的好处是，因为我们其实只是在给原生元素添加样式，所以表单处理的逻辑在基本体验和增强体验里完全一样。

第 16 章

滑　　块

16

滑块控件（slider）的作用是截取连续区间里的某个值或者某段数据范围。许多用户能立刻感觉到滑块的直观性，因为它们的外观和功能模拟了日常用品中的实体滑块，比如立体声音响、汽车仪表盘控制器以及家用电器。它里面的“轨道”把能做的选择限制在可用选项组成的连续区间内，而“手柄”的位置则是对当前选中值的清楚反馈。

通常，Web应用程序里的滑块截取的是价格和音量这样的数值，但是它们同样可以用于几乎所有能构成连续区间值的集合，比如用字母评定的分数等级、金融债券评级或者从积极到消极的情绪。它们可以获取单个值，也能扩展成两个或多个滑块手柄，用来截取多个值或者一段范围。

滑块有下面这些常见用途。

- 当作过滤工具来设置参数，比如产品尺寸、重量或容量；最低价和最高价之间的范围；图集里照片缩略图的显示尺寸；RSS阅读器里的摘要显示长度；地图的缩放等级。
- 嵌入式音频或视频的播放控件。滑块在里面扮演媒体时间线的角色，可拖动的手柄则会自动前进，指明当前的播放位置。

当前，没有哪一种原生的HTML表单控件能够实现跨浏览器的滑块组件。使用滑块的网站通常会标记一组HTML元素（经常是div），然后用CSS和JavaScript来应用样式和滑块行为。有可访问性意识的开发者还会加入基本的键盘支持和指派必要的ARIA属性，以确保现代的屏幕阅读器能够了解该如何向有视力障碍的用户传达意义和行为。

但是，如果用户没有启用JavaScript，或者无法渲染出滑块正常工作所需的CSS样式怎么办？一些大量使用滑块的流行网站（比如旅游网站Kayak就提供了用来过滤航班时间和价格的滑块）只要求用户的浏览器必须支持脚本功能。它们不提供备用的选择，完全没有考虑那些没有启用JavaScript的用户访问这项功能的可能性。

你没有理由把用户锁在关键功能之外。使用渐进增强方法进行构建，就可以从基础标记里的一个标准HTML表单元素（比如文本input或select元素）起步，向基本体验里的所有用户提供核心功能，然后在增强体验里把它转变成一个滑块组件。本章会探索如何创建一个普遍可访问的滑块。

16.1　X光透视

在线房地产搜索网站经常使用滑块来简化目标搜索条件的设置和调整工作。本章的设计范例

会考虑一个带有两组滑块的公寓搜索表单：第一组滑块收集的数值范围是最高价格，以及最少的卧室和卫生间数量；第二组收集的是定性值，这些生活便利设施的相对重要性分成从“不重要”（Not important）到“必须要有”（Must have）4级连续区间，并在滑块右边显示所选值的反馈信息，如图16-1所示。

图16-1　增强体验的目标设计

X光透视这些滑块后，我们发现文本input、select和单选按钮都是很好的基础标记候选元素。判断哪种元素最适合基本体验的方法是分析每个表单字段的具体数据输入要求，并考虑如何平衡以下两点：用户输入的速度和灵活性，以及限制数值在可接受范围内的需求。

前三个滑块收集特定的最小和最大范围内的数值。最大租金（rent）滑块允许填写0~5000美元的任意数值，而卧室（bedroom）与卫生间（bathroom）滑块接受的数值范围就小一些（0~10）。如果我们为最高价字段使用一个HTML select元素，它就需要在标记里列出足足5000个可选项来实现最大的灵活性，即使我们限制价格按100美元递增，仍然会有50个值。这种实现方式从技术上说是可行的，但是从用户体验的角度看很可能不是最佳的做法。文本input则是一种更好的选择，它为租金字段的基础标记提供了大得多的灵活性和用户友好度：在字段里输入数字要比用鼠标或键盘与下拉菜单交互更快，也更容易。为了保持一致性，给卧室和卫生间字段使用文本input。

HTML5规范提供了一个type=number的input属性，它支持使用一些定义最小值和最大值的属性来帮助验证用户输入。相比简单一些的type=text，利用这些属性能更充分地描述租金、卧室和卫生间字段。而且因为HTML3和HTML4规范都规定input元素在没有指定类型（或者是未知类型）时会默认设置成type=text，所以可以安全地在基础标记里使用HTML5的number input，并确信它在不理解number类型的浏览器里仍然会显示为一个完美可用的标准text input，如图16-2所示。

Apartment Criteria
Max Rent ($): 2000
Min bedrooms: 2
Min baths: 2

图16-2　基本体验里用于租金、卧室和卫生间数值的文本输入框

使用text或number类型的input时需要考虑的一个重点问题是数据验证。与select不同，input不提供约束或实时反馈来防止用户输入无效值，比如负数、字母、符号或者在可接受范围以外的值。为了确保在基本体验里输入的数据有效，服务器应当在用户每次提交更新结果的表单时都验证输入，如果不是可接受的值就显示一条错误消息。在增强体验里，可以用JavaScript在前端实现同样的验证逻辑，为有能力的浏览器上的用户提供即时反馈。

我们还应该要考虑用户与滑块控件交互时的数据间隔。一个滑块能获取的值的数量受滑块轨道的宽度限制，在这个设计里大约是300像素。我们的最少卧室与卫生间输入字段只接受1~10的数字，所以滑块的实现不是问题：可以沿着300像素的轨道把10个值简单地均匀分开摆放。

租金字段则是另一回事：它有5000个可接受的值（0~5000美元，按1美元递增），但是滑块的手柄在轨道宽度以内只能提供300个可能的像素位置。如果我们均匀分配这些像素的话，这就意味着要按17美元递增。虽然把手柄增量设为17可能不会给使用鼠标的用户造成任何问题（除了这个数量看似有些随意），但是对那些使用键盘方向键的人来说可能就会又慢又不方便。为了优化滑块，使其能快速选择容易辨认的数字，我们将指定手柄按50美元递增，用户的操作方法既可以是拖动手柄，也可以是使用方向键左右移动。

使滑块和文本input同时出现在界面上，就能让用户选择他们想要的交互方式：移动滑块手柄来指定某个租金值（用于快速但不太精确的选择），或者在文本input字段里输入一个数字（用于精确输入美元金额）。因为这两者都处于收集数据的激活状态，所以我们会确保它们互相同步：移动滑块手柄的位置会更新input的文本值，反之亦然，如图16-3所示。

Max Rent ($): 2507

图16-3　在增强体验里，滑块和文本输入框都可用

生活便利设施偏好（Amenity Preferences）部分的滑块（即表单里第二组的三个滑块）收集的数据与租金、卧室和卫生间字段有着显著区别。在这里，用户可以从以下四个预设值里选择一个：“不重要”（Not important）、“较为重要”（Somewhat important），“相当重要”（Pretty important）和“必须要有”（Must have）。要在基础标记里表现这些选项，可以使用一组单选按钮，因为它能以易于浏览的方式显示出所有可能的选项，并限制用户只能做有效的选择。但是，考虑到各个单选按钮及其标签所需空间，select元素在基本体验具有更加紧凑和易于阅读的格式，还能提供相同的约束。用户做出选择时，被选中的选项会与标签组合在一起，形成一段易于阅读的短语（比如“有地铁站：较为重要”），如图16-4所示。

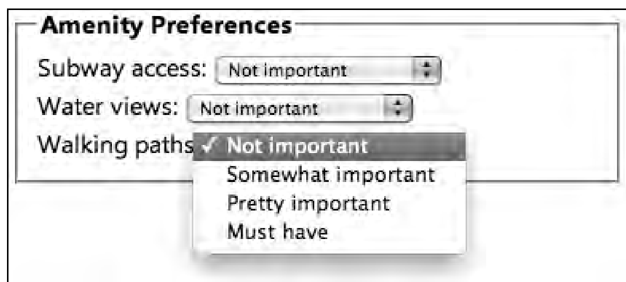


图16-4 基本体验里用于生活便利设施偏好的下拉列表框

让select在增强体验的页面上保持可见没有什么益处，因为滑块能完全取代select的功能。在这个案例里，我们会隐藏原有的select，然后在滑块右边显示当前所选值的文字反馈信息。因为滑块本身也能向屏幕阅读器用户传达自己的值，所以对这些用户我们会用ARIA来隐藏可见的文字反馈信息，以避免重复，如图16-5所示。

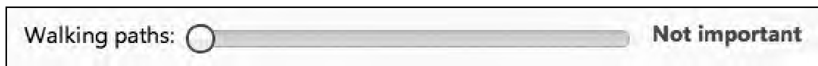


图16-5 增强体验里的select是隐藏的，被文字反馈信息取代

一个真正可用的滑块控件必须能同时顺利地实现鼠标和键盘操作，并且需要精心设计操作逻辑来优雅地处理拖放功能。与其从头创建一个滑块，不如借用jQuery UI库里包含的这个健壮的滑块（http://jqueryui.com/docs/slider）来节省时间和开发投入。这里也会使用它。

对某个input或select元素调用我们的增强脚本时，它会充当类似中间人的角色，解析基础标记并找出配置jQuery UI滑块所需的值。之后，jQuery UI滑块插件会插入增强标记来生成滑块轨道和手柄，应用拖动吸附（drag and snap）行为，根据手柄运动计算值的变化，以及支持键盘操作。

我们的插件脚本会通过添加ARIA支持来完善jQuery UI滑块插件的能力，写作本书时的jQuery UI版本尚未包括这一点（但是快了）。最后，它会作为代理，确保数据值的任何变化都能在增强滑块和原生input与select元素之间传递。

计划好如何处理基本体验里的各种滑块功能之后，就可以着手开发我们的自定义滑块了。

HTML5的input type=range元素：未来的原生滑块

HTML5规范给input元素添加了一个新的type=range属性，用于收集“非精确性”数值（这是W3C对“滑块”的叫法）。最新版的Opera和Webkit浏览器已经能把这个类型的input渲染成一个滑块，因此对这个原生滑块的浏览器支持正在慢慢普及[①]。

① IE 10、Firefox 23、Chrome 4、Safari 3.1、Opera 9、iOS Safari 5、Android 4.2及之后版本的浏览器都支持这个类型。——译者注

从技术上看，你可以安全地在今天的基础标记里指定type=range的input。input元素会在未提供类型或者类型未知时默认使用text类型。你需要注意的仅仅是从设计的角度看，Opera和Safari浏览器渲染range input滑块的方式有着巨大的区别，无论从功能还是外观上来说都是如此。与其说这个元素可以安全使用，倒不如说它感觉更像是一种实验性的元素。我们期待在未来的某一天可以用一致的方式使用这些滑块，如图16-6所示。

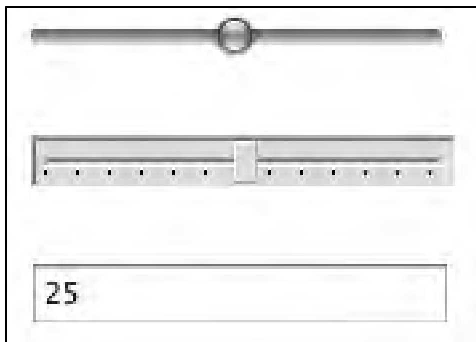


图16-6　最新版的Safari（上）和Opera（中）浏览器为range input渲染出的原生滑块。火狐浏览器（下）则显示出一个标准的文本input，因为它不支持range类型

16.2　创建滑块

构建公寓搜索工具滑块的第一步是创建在基本体验里获取数据值的基础标记。然后，为有能力的浏览器创建用于交互式滑块的增强标记，并编写一些脚本来应用它们的行为，以及让数据在滑块与input元素之间保持同步。

16.2.1　基础标记和样式

对一开始的三个字段，使用数字input元素，对后三个则使用select元素。我们会给它们各自添加关联的label标签，并放置在一个form里以便提交。

16.1节提到，每个数字形式的数据字段（最高租金价格、卧室数量和卫生间数量）都会编写成一个类型为HTML5 type=number的input，并加上min和max属性以及它们的值。我们会给input预先生成一个初始的默认值，并指定一个size属性，这个属性用一种灵活的方式来告诉浏览器应该按照某种宽度显示input，这个宽度需要能够容纳某一特定的字符数量以符合可接受的数据限制条件。

```
<input type="number" name="price" id="price" value="2000" min="0" max="5000" size="4" />
```

为了增加一点额外的意义，利用它们在增强体验里给屏幕阅读器提供更明确的反馈，我们会使用HTML5的data-前缀创建一个名为data-units的自定义属性，用它来指定input中值的单位（在这个案例里是美元）。这样给移动的滑块设置aria-valuenow（当前值）属性后，它就会朗读

出更人性化的“1500美元”，而不仅仅是里面的数值“1500”。我们还会使用反馈文字来填充滑块手柄的title属性，它在大多数浏览器里会被渲染成一条工具提示：

```
<input type="number" name="price" id="price" value="2000" min="0"
max="5000" size="4" data-units="dollars" />
```

为每个input都创建一个标签，并用for属性把它正确关联到带有相匹配id的input上：

```
<label for="price">Max Rent ($):</label>
<input type="number" name="price"  id="price"  value="2000" min="0"
max="5000" size="4" data-units="dollars" />
```

卧室和卫生间数量input的标记也会用同样的方式组建，包括正确关联的标签，以及恰当的min、max、size和data-units值。

三个生活便利设施方面的滑块在基础标记里会用select元素编写，并同样关联label的for属性和select的id，就像这样：

```
<label for="subway">Subway access:</label>
<select name="subway" id="subway">
  <option value="0">Not important</option>
  <option value="1">Somewhat important</option>
  <option value="2">Pretty important</option>
  <option value="3">Must have</option>
</select>
```

这些select滑块里的option值本身已经是完整和可阅读的，也天生带有约束性，所以不需要给它们添加额外的属性。

定义完各个表单字段的基本标记后，我们将在基本样式表里应用一种全局字体，使基本体验看起来更美观一些：

```
body, input { font-family: "Segoe UI", Arial, sans-serif; }
```

16.2.2 增强标记和样式

前面提到，我们会让jQuery UI滑块插件负责生成增强标记，并给标记里所有指定为滑块的元素应用滑块行为。

jQuery UI滑块会生成一个锚链接作为滑块手柄，然后把它放在一个当作轨道的div内。拖动滑块手柄时，脚本会动态设置锚上的内联样式，通过把CSS left属性设置成轨道最大值的某个百分比反映出手柄当前的位置：

```
<div class="ui-slider ui-slider-horizontal">
  <a href="#" class="ui-slider-handle" style="left:40%"></a>
</div>
```

> **注意** 为了清楚起见，我们省略了jQuery UI滑块标记里的一些类，它们的作用是应用某个jQuery UI主题。

我们的增强脚本需要添加许多ARIA属性使滑块可访问。无论基础标记是数字类型的input还是select，这些属性都是一样的。

首先，给body元素添加一个值为application的ARIA role属性，使屏幕阅读器能够识别出这个滑块控件，并遵循我们指派的那些自定义键盘快捷方式：

```
<body role="application">
```

然后，脚本会给充当滑块手柄角色的锚链接指派许多ARIA属性。（因为手柄承担了滑块的所有交互功能，而轨道主要用来提供视觉反馈，所以各种ARIA角色都会放在锚链接上。）

锚元素上一个名为slider的role告诉屏幕阅读器它扮演滑块控件的角色，而不是普通的链接，这样在链接获得焦点时它就会朗读出关于该控件的一段描述：

```
<div class="ui-slider ui-slider-horizontal" role="application">
  <a href="#" class="ui-slider-handle" role="slider" style="left:40%"></a>
</div>
```

利用aria-labelledby属性可以关联滑块与页面上某个描述它的元素。向滑块提供数据并在数据改变时与其同步的原生基础元素（在这个案例里是input元素），在网页里已经有了一个关联的label，这个标签同样能准确地描述滑块。在增强体验里，可以使用input元素的label充当滑块的标签。aria-labelledby属性必须通过id来引用别的元素，因此用JavaScript给标签生成一个唯一的id，方法是使用其for属性的值，然后加上一个-label后缀：

```
<label for="price" id="price-label">Max Rent ($):</label>
```

然后用同一个ID生成滑块锚链接上的aria-labelledby属性，这样就能使它关联label元素，以服务屏幕阅读器用户：

```
<div class="ui-slider ui-slider-horizontal" role="application">
  <a href="#" class="ui-slider-handle" role="slider" aria-labelledby=
"price-label" style="left:40%"></a>
</div>
```

aria-valuemin和aria-valuemax属性标明了滑块控件的最小值和最大值。同样，我们会借用基础标记里的值，对各个input用的是它的min和max属性，select则是它的第一和最后一个值：

```
<div class="ui-slider ui-slider-horizontal" role="application"
style="left:40%"></a>
  <a href="#" class="ui-slider-handle" role="slider" aria-valuemin="0"
  aria-valuemax="5000" aria-labelledby="price-label" style="left:40%"></a>
</div>
```

最后，每当滑块手柄移动或者它关联的表单字段值发生变化，就用脚本来动态更新两个额外的ARIA属性：aria-valuenow（滑块的当前数字值）和aria-valuetext（供屏幕阅读器朗读的人性化数值反馈文字）。这一过程对我们的两个基础元素稍有不同。在基于select的滑块上，将option里的文本填入data-units属性。为了让aria-valuetext对数字类型的input更有用，我们的脚本会给当前的input值（2000）加上基础标记里的data-units属性字符串（“dollars”），以改进听觉反馈（“2000 dollars”）：

```
<div class="ui-slider ui-slider-horizontal" role="application">

  <a href="#" class="ui-slider-handle" role="slider" aria-valuemin="0"
aria-valuemax="5000" aria-valuenow="2000" aria-valuetext="2000 dollars"
aria-labelledby="price-label" style="left:40%"></a>

</div>
```

可以借助一个ARIA属性来改善非屏幕阅读器用户的体验。我们的增强脚本会用aria-valuetext属性里的反馈文字给手柄链接生成一个title属性，它会被渲染成一条描述性的工具提示。（这不是必需的，而是一项“锦上添花”的功能，因为这些滑块input在页面上仍然可见。）

```
<div class="ui-slider ui-slider-horizontal" role="application">

  <a href="#" class="ui-slider-handle" role="slider" aria-valuemin="0"
  aria-valuemax="5000" aria-valuenow="2000" aria-valuetext="2000 dollars"
  aria-labelledby="price-label" title="2000 dollars" style="left:40%"></a>

</div>
```

使用相同的标记结构来创建其余的网页控件滑块。在草拟出具备ARIA功能的增强标记之后，现在编写增强样式，让jQuery UI滑块的外观接近我们的目标设计。

1. 用于增强体验的CSS

首先，给滑块容器添加样式，它在我们的设计里用来给滑块手柄提供一条轨道。它有固定的宽度和高度，一张用来给轨道增加一点渐变质感的背景图像，圆化的边角和一个边框。它还是相对定位的，让滑块手柄和range类型的input元素能它里面绝对定位。

```
.ui-slider {
  position:relative;
  top:.8em;
  float: left;
  width:293px;
  height:1em;
  background:#ebebec url(../images/bg-slider.png) top repeat-x;
  border:1px solid #aaa;
  -moz-border-radius:.5em;
  -webkit-border-radius:.5em;
  border-radius:.5em;
}
```

我们的滑块轨道及其标签如图16-7所示。

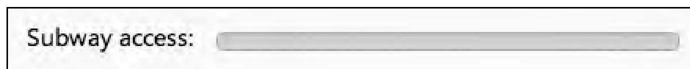


图16-7 添加样式后的滑块轨道

滑块手柄借助负的top和margin-left属性进行绝对定位，使它居中处于轨道上方。我们给它添加样式的方法是使用width和height属性创建一个方块，然后给它一个实线边框、一张背景图像，以及CSS3的border-radius属性来圆化边角，使手柄看起来像一个圆（这个属性并不能得到

所有浏览器的支持，但它能安全地降级）：

```
.ui-slider .ui-slider-handle {
  position:absolute;
  z-index:2;
  width:1.6em;
  height:1.6em;
  top:-.5em;
  margin-left:-.8em;
  cursor:pointer;
  border:2px solid #444;
  background:#fff url(../images/bg-slider-handle.png) 50% repeat-x;
  -moz-border-radius:.9em;
  -webkit-border-radius:.9em;
  border-radius:.9em;
}
```

添加上这些样式后，滑块手柄如图16-8所示。

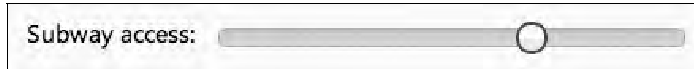


图16-8　添加样式后的滑块手柄和轨道

为了在用户鼠标悬停或利用键盘使滑块手柄获得焦点时创建视觉反馈，给hover和focus状态应用一张不同的背景图像。在激活状态下（点击或着按键），我们会移除背景图像，留下平白的外观。

```
.ui-slider .ui-slider-handle:hover, .ui-slider .ui-slider-handle:focus {
  background-image:url(../images/bg-slider-handle-hover.png);
}
.ui-slider .ui-slider-handle:active {
  background-image:none;
}
```

对最大租金这样的数字型input滑块，我们会利用jQuery UI滑块里的range-min选项。这个选项会用颜色填充从滑块轨道左边缘到手柄当前位置这一段范围（就像一个温度计），使这部分突出显示，从而在视觉上提示用户他们正在选择所选值以下的所有租金价格或卧室数量，而不仅仅是连续区间里的单个点。启用这个选项后，插件会给滑块脚本添加一个额外的div元素，并动态设置一个内联的width百分比样式来匹配手柄的位置：

```
<div class="ui-slider ui-slider-horizontal">
  <div class="ui-slider-range ui-slider-range-min" width="42%"></div>
  <a href="#" class="ui-slider-handle" style="left:42%"></a>
</div>
```

我们会给这个范围div添加深色背景图片的样式，使滑块轨道看起来像一直填色到手柄的位置：

```
.ui-slider .ui-slider-range {
  position:absolute;
  z-index:1;
  font-size:.7em;
  display:block;
  border:0;
```

```
  top:0;
  height:100%;
  background:#999 url(../images/bg-slider-range.png) 50% repeat-x;
  left:0;
}
```

我们的数字型范围滑块现在就有了一段填充色，让人能更清楚地看出选中的是一个最大值，它下面的所有值都在选择范围之内。

给数字型input添加样式使它们看上去可编辑，因为除了滑块，用户也可以使用它们，如图16-19所示。在源代码顺序里，滑块的div处于input前面，将两者设置成float:left会让各个滑块出现在其对应input的左侧。把font-weight设置为bold，指定边框的颜色，并使用CSS3的border-radius属性来圆化input的边角：

```
input#price, input#bedrooms, input#baths {
  float:left;
  font-size:1.3em;
  font-weight:bold;
  width:55px;
  border:1px solid #ccc;
  -moz-border-radius:3px;
  -webkit-border-radius:3px;
  border-radius:3px; padding:.2em;
}
```

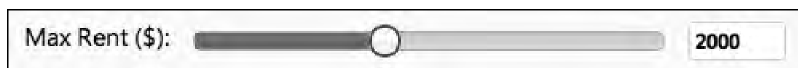


图16-9 数字型滑块带有一段从轨道左端到手柄的填充色，右边放置了数字型input

对三个生活便利设施滑块，用CSS把它们的select元素隐藏到屏幕之外，同时在网页中保留它们用来提交表单。我们的增强脚本会给这些元素指派ARIA的aria-hidden=true属性，从而更好地支持具备ARIA功能的屏幕阅读器，告诉它们这些select元素是隐藏的。

```
select#subway, select#water, select#walking {
  display: none;
}
```

为了给生活便利设施滑块里的值提供视觉反馈，脚本会在滑块后面放一个带有slider-status类的反馈div，然后用当前的值动态更新它。给这个div应用一个值为true的aria-hidden属性，让它对屏幕阅读器隐藏，因为反馈文字和滑块本身的aria-valuetext属性信息是重复的。

```
<div class="slider-status" aria-hidden="true">Pretty important</div>
```

将这些反馈div元素与滑块一起添加样式，类似于给一开始的三个滑块添加input元素样式的做法，如图16-10所示。

```
.slider-status {
  float:left;
  font-weight:bold;
  font-size:1.5em;
  color:#444;
}
```



图16-10　添加样式后的生活便利设施滑块，带有当前选择的反馈信息

最后，在生活便利设施滑块的上方添加一个图例，说明选择范围是从不重要到必须要有，如图16-11所示。

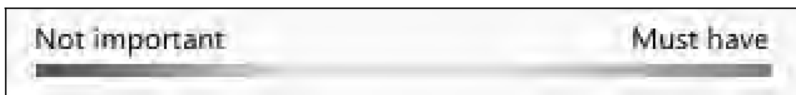


图16-11　生活便利滑块的图例

用JavaScript插入图例的标记。它的文字内容由第一个select菜单（因为所有生活便利设施滑块的选项都相同，它也不例外）里的第一和最后一个文本值生成，然后再给它添加一张背景图像来形成颜色渐变区间：

```
<div class="sliders-labels">
  <span class="label-first">Not important</span>
  <span class="label-last">Must have</span>
</div>
```

图例标记的作用只是提供视觉反馈，所以应用一个值为true的aria-hidden属性，告诉屏幕阅读器这个div没有意义。

```
<div class="sliders-labels" aria-hidden="true">
  <span class="label-first">Not important</span>
  <span class="label-last">Must have</span>
</div>
```

最后，我们会给图例添加样式，让它更像我们的目标设计：

```
.sliders-labels {
  height:2em;
  margin-left:145px;
  width:293px;
  background:url(../images/bg-continuum.png) bottom left no-repeat;
  padding-top:1em;
}
.sliders-labels span.label-first {
  float:left;
}
.sliders-labels span.label-last {
  float:right;
}
```

16.2.3　滑块脚本

用jQuery UI的滑块插件来生成实际的滑块控件，因此增强脚本会扮演桥梁的角色，负责将基础标记解析成滑块能够使用的格式，管理ARIA属性的添加和更新，并确保滑块与input或select

的值在任何时候都保持同步。网页则需要同时引用jQuery和jQuery UI这两个库。

注意 jQuery UI库和滑块这样的组件专用插件可以在http://jqueryui.com/download上下载。

我们的脚本在处理数字型input滑块和定性的select滑块时使用不同的方法。这里会介绍处理每种滑块的步骤。

1. 生成基于input的滑块

为了根据数字型input元素生成滑块，增强脚本要做的第一件事是遍历网页上的各个input元素，收集生成滑块所需的标签、值和其他信息：

```
$('#price, #bedrooms, #baths').each(function(){
  //这里放的是用于每个input的代码……
});
```

在each循环里，脚本首先会生成对循环里当前input的引用，然后给label元素生成并应用一个id，保存来自data-units属性的单位名称，并创建一个变量来储存人性化的反馈文字，这些文字会用作ARIA和滑块手柄的工具提示：

```
//保存对input元素的引用
var input = $(this);

//找到关联的label元素
var thisLabel = $('label[for=' + input.attr('id') + ']');

//使用它的for属性 + 'label'来生成它的唯一ID
thisLabel.attr('id', thisLabel.attr('for') + '-label');

//从自定义的data属性里获取input的单位
var thisUnits = input.attr('data-units');

//获得input的值，创建一个包含单位的人性化版本
var friendlyVal = input.val() + ' ' + thisUnits;
```

为了确保屏幕阅读器用户能用键盘控制滑块，给body元素应用一个名为application的role：

```
$('body').attr('role','application');
```

接下来，脚本会添加一个div，供jQuery UI插件插入滑块的标记：

```
var slider = $('<div></div>');
```

这个div会被插入到网页中，位置是input之前。（通常我们会在创建出控件之后再做这一步，但是滑块插件要求网页里先有标记，然后才能进行配置。）

```
slider.insertBefore(input);
```

然后就可以初始化各个滑块，传递键/值对形式的选项到jQuery UI滑块的API，选项之间用逗号分隔。设置min和max选项，值分别来自各个input的min和max属性。滑块的初始值用value选项进行设置，该值从input的value里获取并解析成一个数字。range选项可以设置一段视觉高亮，具

体范围可以是两个滑块手柄之间，也可以是从某个手柄到滑块的末端。我们把范围设成min，这是告诉插件要创建一个从滑块起点到手柄的范围元素。最后，step选项设置了滑块手柄移动的增量，我们会使用一种称为三元运算符（ternary operator）的特殊条件语句，如果当前input的id是price，就把step选项设为50，否则就设为1：

```
slider.slider({
  min: input.attr('min'),
  max: input.attr('max'),
  value: parseInt(input.val(),10),
  range: 'min',
  step: input.is('#price') ? 50 : 1
});
```

根据这些信息，jQuery UI插件会创建出功能完整的滑块控件，并将它插入文本input之前的div中。

滑块能正常工作之后，给它添加ARIA属性，使它对屏幕阅读器用户更具有意义，也更可用：

```
slider
  .find('a')
      .attr({
          'role': 'slider',
          'aria-valuemin': input.attr('min'),
          'aria-valuemax': input.attr('max'),
          'aria-valuenow': input.val(),
          'aria-valuetext': friendlyVal,
          'aria-labelledby': thisLabel.attr('id'),
          'title': friendlyVal
    });
```

各个滑块更新时也应该同步更新原来的input元素。我们会利用滑块插件的自定义slide事件，每当滑块的当前值发生变动时就用它来更新input：

```
//给滑块绑定一个自定义的slide事件（slide是jQuery UI里的一个事件）
slider.bind('slide', function(e, ui){
  //设置input的值为滑块的值
  input.val(ui.value);

  //得到人性化的值
  friendlyVal = input.val() + ' ' + thisUnits;

  //更新滑块手柄的属性值
  slider.find('a').attr({
      'aria-valuenow': input.val(),
      'aria-valuetext': friendlyVal,
      'title': friendlyVal
  });
});
```

我们还想让input能反向通知滑块控件，在用户给input字段输入某个值后更新滑块手柄的位置。为此，再次使用这个插件的slider方法。我们提供的第一个参数是要更新的滑块选项（value），第二个选项则是input的当前值：

```
input.keyup(function(){
  slider.slider('value', parseInt($(this).val(),10));
});
```

现在，我们有了一个根据标准input构建的滑块，以及一套让两者保持同步的方法。提交表单时，各个input的值随表单数据发送的方式在基本体验和增强体验里都是相同的。

2. 生成基于select的滑块

通过解析基础HTML来初始化select滑块组件的基本步骤和之前的input非常相似，但是select用到的方法和语法稍有不同。

第一步是遍历各个需要生成滑块的select：

```
$('#subway, #water, #walking').each(function(){
  //用于各个select元素的代码
});
```

在each循环里，脚本会创建一个变量来引用当前的select元素，给这个select应用aria-hidden=true，给label元素生成并应用一个id，获得用于当前所选option的文本，然后创建一个用来容纳滑块的div：

```
//引用当前的select元素
var select = $(this);

//对启用ARIA的屏幕阅读器隐藏该select元素
select.attr('aria-hidden','true');

//保存对关联标签的引用
var thisLabel = $('label[for=' + select.attr('id') + ']');

//使用它的for属性 + `-label`来生成它的唯一ID
thisLabel.attr('id', thisLabel.attr('for') + '-label');

//找到人性化的值
var friendlyVal = select.find('option')
          .eq( select[0].selectedIndex )
          .text();

//创建制作滑块所需的div
var slider = $('<div></div>');
```

滑块的div会插入到select元素之前：

```
slider.insertBefore(select);
```


脚本然后会在div上调用jQuery UI的slider方法，传递最小值、最大值和当前值这三个配置选项来创建一个滑块。min和max选项指的是select选项的第一个和最后一个序号（从0开始）。当前值的获取方法是检查原始select元素里选中的option。将step选项设为1，因为select里的每一个option都必须体现在滑块里。

```
slider.slider({
  min: 0,
  max: select.find('option').length-1,
```

```
    value: select[0].selectedIndex,
    step: 1
  });
```

我们给这个滑块添加的ARIA属性与之前给input滑块添加的属性相同，这些属性让屏幕阅读器能够访问滑块：

```
slider
  .find('a')
  .attr({
      'role': 'slider',
      'aria-valuemin': 0,
      'aria-valuemax': select.find('option').length-1,
      'aria-valuenow': select[0].selectedIndex,
      'aria-valuetext': friendlyVal,
      'title': friendlyVal,
      'aria-labelledby': thisLabel.attr('id')
 });
```

因为select被隐藏了，所以接下来插入一个带有当前值的div作为视觉反馈。我们还会绑定（bind）插件的slide事件来监视滑块手柄位置的变化，并在滑块值发生变动时做两件事：更新原始select上的选中值，然后用人性化的值来更新反馈div里的文本。还会应用aria-hidden属性防止屏幕阅读器用户重复获取可以在select的ARIA属性上得到的值。

```
//在滑块后面追加状态div
slider.after('<div class="slider-status" aria-hidden="true">'+friendlyVal + '</div>');

//绑定slide事件
slider.bind('slide', function(e, ui){
      //设置下拉菜单的值来匹配滑块
      select[0].selectedIndex = ui.value;

      //得出人性化的值
      friendlyVal = select.find('option')
                .eq( select[0].selectedIndex ).text()

      //填入反馈div
      slider.next().text(friendlyVal);

      //给滑块手柄应用属性
      slider.find('a').attr({
          'aria-valuenow': select[0].selectedIndex,
          'aria-valuetext': friendlyVal,]
          'title': friendlyVal
      });
});
```

最后，脚本会插入所需的静态标记来构建范围图例的图像和标签，这些图例用颜色进行编码：

```
// 生成图例标记
$('<div class="sliders-labels" aria-hidden="true"><span class="label-first">'+
  $('#subway option:first').text() +
  '</span><span class="label-last">'+
```

```
$('#subway option:last').text() +
'</span></div>')
    // 插入到第一个滑块之前
    .insertBefore('label[for=subway]');
```

16.3 使用滑块脚本

本书附带的滑块演示和代码（位于www.filamentgroup.com/dwpe）包含了jQuery.peSlider.js，这个脚本把本章概述的渐进增强滑块功能包装成一个可以重复使用的插件。peSlider脚本是jQuery UI滑块插件的一个“包装器”，它同时依赖于jQuery和jQuery UI库。

要在你的网页里使用这个脚本，请下载并引用滑块演示页里列出的那些文件，然后在某个数字型`input`或`select`元素（以及两者的任意组合）上调用`peSlider`方法。利用这个插件，用一条jQuery语句就能创建演示页里的所有滑块，比如下面这句：

```
$('#price, #bedrooms, #baths, #subway, #water, #walking').peSlider();
```

调用`peSlider`方法会创建一个jQuery UI滑块，它具备所有的数据和代理逻辑来同步原生元素到滑块上，还有自动计算的默认增量，以及所有用于可访问性的ARIA属性。

在上面的例子里，有3个自定义选项默认不包括在jQuery UI滑块插件和peSlider插件里：给滑块轨道添加高亮填色来显示选中区域的`range`选项；设置自定义增量的修改版`step`选项；在`select`滑块右侧显示当前选中值文字反馈的`div`。针对这些具体的示例，需要给`peSlider`方法传递配置选项，或者用jQuery手动生成某些标记。下面的例子会逐步展示如何用`peSlider`分别实现这些要求。

最前面的三个`input`滑块使用范围反馈元素，它突出显示滑块起点到手柄之间的轨道。我们的peSlider插件封装了原生的jQuery UI滑块，因此可以传递任何原生的jQuery UI滑块选项，比如以键/值对的形式传递创建范围反馈的选项。举个例子，我们会指定`range min`选项：

```
$('#price, #bedrooms, #baths').peSlider({range: 'min'});
```

可以用这种方式传递任何jQuery UI滑块选项。请记住，`min`、`max`、和`value`选项由插件根据`input`或`select`标记自动生成，无需把它们作为参数传递进来。

peSlider插件还会测算出一个合理的默认增量，方法是用滑块的最大值除以它的宽度，然后自动将滑块的`step`选项设置成这个增量。可以很容易地用自定义增量覆盖这个默认值，方法是设置一个具体的`step`值：

```
$('#price').peSlider({step 50});
```

在我们的X光范例里，基于`select`的滑块会更新滑块右侧的一个只读反馈文字区。peSlider插件不会自动进行生成和更新反馈`div`这一过程，因为不是所有的场合都需要用到这种特定的设计元素。不过，这项功能可以使用jQuery实现，做法是调整滑块的`slide`事件来检查新值和更新反馈`div`。

下面的代码会在各个`select`滑块之后插入一个反馈`div`，并使用滑块的`slide`事件，在滑块的

值发生变化时更新反馈div。至于反馈div里的文本，我们只需抓取滑块手柄的aria-valuetext值，因为它已经被格式化成一种人性化的滑块值版本：

```
$('select')
  //插入滑块的反馈div
  .after('<div class="slider-status" aria-hidden="true"></div>')
  //创建滑块
  .peSlider({
      //给滑块的slide事件绑定一个回调函数
      slide:function(){
          //根据滑块元素找到状态div
          var statusDiv = $(this).next().next();

          //找到滑块手柄
          var sliderHandle = $(this).find('a:eq(0)');

          //用aria-valuetext属性设置statusDiv里的文本
          statusDiv.text(sliderHandle.attr('aria-valuetext'));
      }
  })
  .each(function(){
      //根据select元素找到状态div
      var statusDiv = $(this).next();

      //根据select元素，找到滑块手柄
      var sliderHandle = $(this).next().find('a:eq(0)');

      //用aria-valuetext属性设置statusDiv里的文本
      statusDiv.text(sliderHandle.attr('aria-valuetext'));
  });
```

peSlider插件被设计成非常简单的形式，因为jQuery UI滑块提供了大量的功能，我们可以轻松地利用这些功能来扩展它。

注意　要了解jQuery UI滑块里各种可用功能的更多信息，包括竖直方向、额外的回调事件，以及用ThemeRoller.com实现视觉主题，请阅读http://jqueryui.com/demos/slider里的文档。

让滑块更进一步

有两种高级功能可以添加到滑块中，使它具备更高的可用性和交互性。

- 滑块手柄附近的工具提示（无论是在滑块事件中显示还是始终显示）能够提供对当前选中值的反馈。
- 沿着滑块轴的刻度线和标签能提供额外的视觉上下文信息，帮助用户理解选项连续区间里的粒度级别。

我们在自己网站上的一篇实验室文章里探索了如何扩展jQuery UI滑块来添加这两种功能（http://t.cn/zRlHurf）。

本章阐述了如何用渐进增强技巧构建出滑块等非常具有交互性的元素，并使它能在所有设备上正常工作。我们认为，用脚本从基础标记里解析数据并生成增强的JavaScript组件是一种非常强大的方法，它不仅利用了Web 2.0的特性，而且仍然提供了能在所有设备上正常工作的等价语义化HTML。

我们还演示了如何封装现有的、功能完备的插件，使它提供渐进增强的能力和ARIA可访问性支持。如果你想得到某个插件，其中包含你想要的功能，那么我们觉得这就是个好办法。你可以加入必要的可访问性考量，而不必重新发明轮子，从头重新编写一个插件。

第 17 章

下拉菜单

原生的select元素看起来十分简单，但其实是健壮而复杂的组件：它们包含了许多交互功能，比如用于导航和选择选项的键盘映射、单字母输入提示以及鼠标滚轮支持（仅举几例）。

经常会有一些设计要求下拉菜单具有自定义的观感。理想的情况下，可以只用CSS来为一个标准select元素添加样式，同时保留Web用户所期望的那些原生功能。但是，浏览器对给这些元素添加样式所提供的支持非常有限：样式由各种浏览器自行定义（包括文本样式和大小、盒轮廓线和箭头样式、下拉菜单选项格式，以及滚动条），大多数无法用CSS进行配置。

许多网站设计（包括我们设计的一些）基于视觉和功能的原因需要包含自定义的下拉菜单。常见的改进包括：

- 修改select元素文本内容、盒外形或下拉箭头的外观样式来匹配网站独特的视觉设计；
- 在各个菜单项的边上添加图标来强化内容的选择，比如在电子商务网站里显示产品的颜色，或者在国家位置或货币选择页上显示国旗；
- 将选项菜单的格式调整为层级结构，或者把option文本的格式调整为多行，并带有多种文本样式（比如混合粗体与普通字体）。

无论从用户体验还是企业品牌的角度看，追求这些目标都有着正当的理由，而一个自定义的select能够实现这些目标，前提是实现方式正确。

本章会分析如何把一个原生的HTML select元素转变成带有图标和高级样式格式的自定义下拉菜单组件，并详细介绍如何实现你期望从原生select中获得的全部交互功能和可访问性。然后，讨论如何将这些原则扩展到其他各种自定义的形式上（包括复杂的选项内容格式化和单字母输入提示支持），并展示如何将一个健壮的自定义下拉菜单组件集成到你的项目里。

17.1 X 光透视

假设我们正在构建一个零售应用程序，购物者可以在结账过程中选择他们所购商品的礼物包装。我们希望能为用户提供这样的下拉菜单：下拉菜单选项具有良好的视觉反馈，因此添加一张缩略图，里面展示了各种样式的颜色和图案（见图17-1）。与此同时，我们还想定制下拉菜单的字体风格和视觉样式，以匹配整个零售网站的设计。

通过X光透视，可以看出这个自定义下拉菜单有3个元素：一个用于标识控件的标签，一个

显示当前选中项反馈信息的可点击元素，以及一张在它下面展开的选项菜单，里面显示出可用的选项。

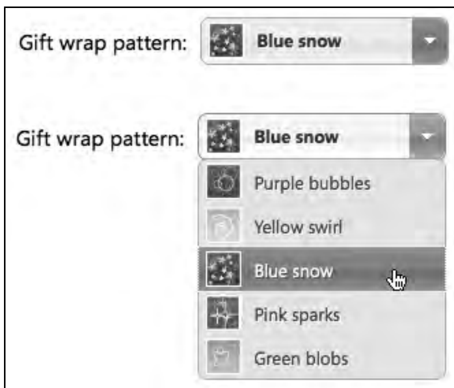


图17-1　自定义下拉菜单的目标设计，每个选项边上都带有图标

这个目标设计体现出标准select元素的所有视觉线索（按钮形状和轮廓线、渐变质感、指示箭头），因此在基础标记里很明显应该从一个原生的select元素起步。基本体验只会显示这个select，再加上内容为Gift wrap pattern:（礼物包装图样：）的label和各个选项。我们认为让用户看一张图能帮助他们做出明智的决定，因此在select后面加上一个链接，连接到显示礼物包装选项预览的网页，如图17-2所示。



图17-2　基本体验里的礼物包装选择工具是一个标准的select控件

某个浏览器通过能力测试后，用JavaScript获取这个select元素的“数据”（标签、类、属性和选项文本），用这些信息动态构建一个自定义下拉菜单组件，它会取代原生的控件，并以一种“代理”的形式与原生select元素在后台通信。

可点击元素的标记必须使用一个原生能获得键盘焦点的元素，可以使用一个标准的锚链接，一个type为button的input，或是一个button元素。应用增强信息时，用JavaScript重写此元素的主要功能（链接到另一张网页或资源，或者提交表单数据），所以选择的依据就成了哪种元素最容易修改样式。可以不使用input按钮，因为它不支持背景图像（我们的设计里有多处悬停状态和礼物包装图标会用到背景图像）。这样就剩下了button和锚元素，无论哪一种都可以有效地用于目标设计。这个范例会使用一个锚元素。

自定义下拉菜单由一张带有样式的链接列表组成。最接近这种结构的语义匹配是一张无序列表（ul），里面用一个内含锚元素的列表项（li）来代表各个菜单项。

目标设计还为每个礼物包装选项都指定了一个独一无二的图标。许多开发者会在增强脚本里加入各个选项的背景图像引用。不建议使用这种方式，因为它限制我们的脚本只能用于这一组数据。如果需要添加额外的自定义下拉菜单组件，就必须为每个组件都复制和定制一个脚本。这样使用JavaScript效率低下，而且难以维护，尤其是在数据（在这个案例里是礼物包装设计）会发生变化的情况下。

与之相反，我们会用基础标记里的属性来"编码"增强体验所需的信息。给每个option都添加一个用来指定图标样式的class属性，然后用增强脚本把这些类转换成增强标记里的对应列表项，并使用CSS设置各个选项的背景图像。这种做法虽然会给基础标记增加额外的属性，但却是无害的（class属性不会影响到option元素的使用或显示方式），利用它就能以一种通用的方式编写脚本，将它用于应用程序里的多个下拉菜单。

自定义下拉菜单看上去会非常像标准的select元素，因此用户可能会期望它能以相同的方式对鼠标点击和键盘命令作出反应。用户应能使用Tab键进入和离开自定义下拉菜单，用鼠标点击或者Tab键进入后按下空格键来打开菜单，用方向键在选项列表里上下移动，以及点击鼠标或按下空格/回车键来选择某个选项。为了在自定义下拉菜单里启用这种行为，需要编写标记和脚本逻辑来改变锚元素的原生行为，使它们转而遵循原生select的行为模式。

对屏幕阅读器来说，构建的自定义下拉菜单应该是一个自定义的下拉菜单组件，而不是一个锚链接与一张无序列表。为此，我们会添加ARIA属性，标明自定义下拉菜单各个部分所扮演的角色、它们之间的关系，以及它们当前的状态和值。

最后，需要确保用户对礼物包装的选择会随表单一起提交。在增强体验里，我们会在网页上保留原生的select元素，但对用户隐藏。只要自定义下拉菜单组件的值发生变化，我们就会把它的值代理发送到原生的select元素上，使这个用于表单提交的元素能保持同步。

17.2　创建可访问的自定义下拉菜单

对这个自定义下拉菜单，创建用于基本体验的基础标记结构，并在这个结构中编写增强脚本所需的属性，从而构建自定义组件。然后，分步介绍如何标记自定义下拉菜单，如何指派向屏幕阅读器用户提供反馈所需的ARIA属性，以及如何给它添加样式来匹配我们的目标设计。最后，分析增强脚本如何生成自定义下拉菜单，将它插入页面，给它应用行为（包括键盘支持和输入提示功能），以及创建隐藏select元素的代理，让表单数据在基本体验和增强体验里能以相同的方式提交。

17.2.1　基础标记和样式

基础标记由一个标准select元素和一个label配对组成。label元素的for属性匹配select的id，它为屏幕阅读器提供元素的关联信息，并让用户可以通过点击label使select获得焦点：

```
<label for="giftwrap">Gift wrap pattern:</label>
<select name="giftwrap" id="giftwrap">
  <option value="pattern-bubbles">Purple bubbles</option>
  <option value="pattern-swirl">Yellow swirl</option>
  <option value="pattern-snow" selected="selected">Blue snow</option>
  <option value="pattern-sparks">Pink sparks</option>
  <option value="pattern-blobs">Green blobs</option>
</select>
<a href="giftwrap-patterns.html" id="giftwrap-patterns">view samples</a>
```

接下来，给select里的每个option值都编码一个唯一的class，我们会在增强体验里借助它们用JavaScript构建自定义下拉菜单选项，并应用图标。

```
<select name="giftwrap" id="giftwrap">
  <option class="purple-bubbles" value="pattern-bubbles">Purple bubbles</option>
  <option class="yellow-swirl" value="pattern-swirl">Yellow swirl</option>
  <option class="blue-snow" value="pattern-snow" selected="selected">Blue snow</option>
  <option class="pink-sparks" value="pattern-sparks">Pink sparks</option>
  <option class="green-blobs" value="pattern-blobs">Green blobs</option>
</select>
```

对基本体验而言，给body和select元素都指派一个font-family属性（这里有必要直接针对select元素，因为一些浏览器不允许表单元素继承字体样式）：

```
body, select { font-family: "Segoe UI", Arial, sans-serif; }
```

应用少量CSS后，基本的select如图17-3所示。

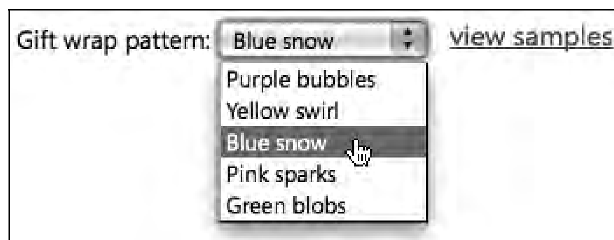


图17-3　应用安全样式后的原生HTML select，以及一个指向礼物包装图像的链接

这个基本体验具有完整的功能，指向样本图像页的链接则添加了出现在增强体验里的那点缺少的内容。现在，我们已经准备就绪，可以开始组装自定义下拉菜单的标记和样式了。

17.2.2　增强标记和样式

讨论组件标记之前，首先需要指导屏幕阅读器识别增强标记里的应用程序控件，这样就能利用自定义的键盘行为。为此，给body元素添加一个值为application的ARIA role属性：

```
<body role="application">
```

接下来，JavaScript会根据基础select元素里的值生成自定义下拉菜单的两个部件（菜单按钮和选项列表）。脚本会把菜单按钮插入到页面标记里的基础select元素（它在整个自定义下拉菜

单就位后会被隐藏）之后，并把选项列表放在网页末尾，以确保它的菜单能可靠地排列在其他绝对定位的网页元素之上。

这个按钮是一个锚链接，由选中选项的文本生成，而选项菜单是一张无序链接列表，由所有选项的文本生成。两者都会被指派一个独一无二的id，所依据的是select元素的id（giftwrap）：

```
<a href="#" id="giftwrap-button">Blue snow</a>
<ul id="giftwrap-menu">
  <li><a href="#">Purple bubbles</a></li>
  <li><a href="#">Yellow swirl</a></li>
  <li><a href="#">Blue snow</a></li>
  <li><a href="#">Pink sparks</a></li>
  <li><a href="#">Green blobs</a></li>
</ul>
```

注意　虽然自定义菜单的行为会由JavaScript处理，但只要有可能，利用内建于HTML元素的原生行为都物有所值。因此，我们在各个列表项里使用的是锚链接，因为锚点在各种浏览器里都是原生可获得焦点的。

给这两个部件指派一些类，稍后用CSS来给它们添加样式：custom-select类会让链接的样式看起来就像一个可点击的按钮，而应用到ul上的custom-select-menu类会让它看起来像个下拉菜单。菜单在网页载入时应该是隐藏的，因此这张列表还带有一个类（custom-select-menu-hidden）：

```
<a href="#" class="custom-select" id="giftwrap-button">Blue snow</a>
<ul class="custom-select-menu custom-select-menu-hidden" id="giftwrap-menu">
  <li><a href="#">Purple bubbles</a></li>
  <li><a href="#">Yellow swirl</a></li>
  <li><a href="#">Blue snow</a></li>
  <li><a href="#">Pink sparks</a></li>
  <li><a href="#">Green blobs</a></li>
</ul>
```

当脚本生成这些元素的标记时，它会遍历原来的select元素并记录选中的选项（如果没有选项被标记为已选中，就记录第一个选项），然后获取分配给option元素的那些礼物包装类，并将它们分配给增强标记，从而给菜单链接和列表选项应用正确的图标：

```
<a href="#" class="custom-select blue-snow" id="giftwrap-button">Blue snow
</a>
<ul class="custom-select-menu custom-select-menu-hidden" id="giftwrap-menu">
  <li class="purple-bubbles"><a href="#">Purple bubbles</a></li>
  <li class="yellow-swirl"><a href="#">Yellow swirl</a></li>
  <li class="blue-snow selected"><a href="#">Blue snow</a></li>
  <li class="pink-sparks"><a href="#">Pink sparks</a></li>
  <li class="green-blobs"><a href="#">Green blobs</a></li>
</ul>
```

为了让锚元素看起来像个菜单按钮，脚本会生成两个span标签，并将它们插入到锚链接里：第一个是custom-select-status，它给选中项添加按钮文字样式；第二个是custom-select-

button-icon，它用于后盖的箭头图标：

```
<a href="#" class="custom-select blue-snow" id="giftwrap-button">
  <span class="custom-select-status">Blue snow</span>
  <span class="custom-select-button-icon"></span>
</a>
```

我们应该通知屏幕阅读器这个链接是一个下拉菜单按钮，而不是一个简单的超链接，因此添加一些ARIA属性：role="button"用于通知屏幕阅读器此链接扮演的是按钮控件的角色，haspopup=”true”用于关联按钮和弹出内容（在这个案例里是一张选项菜单），aria-owns="giftwrap-menu"的作用是将它明确关联到无序列表的ID上：

```
<a href="#" class="custom-select blue-snow" id="giftwrap-button"
role="button" aria-haspopup="true" aria-owns="giftwrap-menu">
  <span class="custom-select-status">Blue snow</span>
  <span class="custom-select-button-icon"></span>
</a>
```

给按钮添加一个内含“ select”（即“下拉菜单”，请注意此单词前的空格）文字的span，我们会让它对视力正常的用户隐藏。这段文字会作为额外的角色描述，用于那些不完全支持ARIA属性的屏幕阅读器。（举个例子，写作本书时，苹果公司的VoiceOver屏幕阅读器支持button角色，但不会通知用户某个元素带有haspopup="true"属性，因此扮演的是菜单按钮的角色。这个span到位后，VoiceOver就会在菜单按钮获得焦点时朗读出“Blue snow select button”，即“蓝雪下拉菜单按钮”。）

```
<a href="#" class="custom-select blue-snow" id="giftwrap-button"
role="button" aria-haspopup="true" aria-owns="giftwrap-menu">
  <span class="custom-select-status">Blue snow</span>
  <span class="custom-select-button-icon"></span>
  <span class="custom-select-roletext"> select</span>
</a>
```

同样，给选项列表添加一些ARIA属性，让屏幕阅读器理解它是自定义下拉菜单的一部分。给ul指派一个值为listbox的role来标明它是下拉菜单项的容器，然后指派aria-hidden="true"来告诉屏幕阅读器选项列表当前是隐藏的。用一个指向菜单按钮ID（giftwrap-button）的aria-labelledby属性来明确关联选项列表和菜单按钮：

```
<ul class="custom-select-menu custom-select-menu-hidden" id="giftwrap-menu"
role="listbox" aria-hidden="true" aria-labelledby="giftwrap-button">
 ...
</ul>
```

我们会给每个选项链接都添加一个role="option"属性来标明它是可选择的，aria-selected属性则标明了选项的状态：

```
<ul class="custom-select-menu custom-select-menu-hidden" id="giftwrap-menu"
role="listbox" aria-hidden="true" aria-labelledby="giftwrap-button">
  <li class="purple-bubbles">
  <a href="#" role="option" aria-selected="false">Purple bubbles</a>
 </li>
 ...
</ul>
```

注意　这个脚本会在用户与菜单交互时动态更新所有的aria-hidden和aria-selected属性，还会在后台更新基础select，让它的选中项始终匹配自定义菜单。

我们还会给每个选项链接都添加tabindex属性。在默认情况下，页面上的所有链接都能通过Tab键导航获得焦点。但在这个案例里，我们想让菜单被视为自定义下拉菜单组件的一部分，通过点击菜单按钮（而不是网页上的一列链接）进行访问，因此把各个选项链接移出制表键顺序，方法是指派一个值为-1的tabindex。

```
<ul class="custom-select-menu custom-select-menu-hidden" id="giftwrap-menu"
role="listbox" aria-hidden="true" aria-labelledby="giftwrap-button">
  <li class="purple-bubbles">
      <a href="#" tabindex="-1" role="option" aria-selected="false">Purple
      bubbles</a>
  </li>
 ...
</ul>
```

增强标记结构到位后，给它添加样式来匹配目标设计。

菜单按钮看起来像一个占据一定面积的可点击物体，因此我们会让它向左浮动（这使它具备块级属性），给它指定一个宽度值，然后为支持圆角和文字阴影的浏览器添加border-radius和text-shadow属性。再用边框和渐变背景图像进一步修饰它：

```
.custom-select {
  float:left;
  text-decoration:none;
  text-align:left;
  cursor:pointer;
  position:relative;
  background:#fff url(../images/button-silver.gif) left center repeat-x;
  border:1px solid #b3b3b3;
  font-size:1.3em;
  font-weight:bold;
  overflow:visible;
  -moz-border-radius:7px;
  -webkit-border-radius:7px;
  border-radius:7px;
  width:180px;
  text-shadow:1px 1px 0 #fff;
}
```

菜单按钮内部的两个span（分别用于当前选中项文本和带箭头图标的菜单后盖）会设置成向左浮动。给后盖指定一张背景图像并加上圆角：

```
.custom-select-status {
  float:left;
  line-height:2;
  color:#444;
  padding:.2em 8px;
}
```

```
.custom-select-button-icon {
  float:right;
  background:#fff url(../images/button-green.gif) left center repeat-x;
  height:2.5em;
  width:2em;
  -moz-border-radius-topright:7px;
  -webkit-border-top-right-radius:7px;
  border-top-right-radius:7px;
  -moz-border-radius-bottomright:7px;
  -webkit-border-bottom-right-radius:7px;
  border-bottom-right-radius:7px;
}
```

隐藏基础标记里的View Samples（查看示例）链接，因为不再需要把它放在自定义下拉菜单边上了，如图17-4所示。

```
#giftwrap-patterns { display: none; }
```

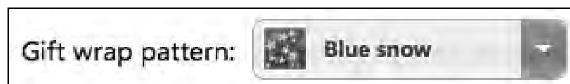


图17-4 应用样式后的自定义下拉菜单按钮

悬停和聚焦状态（通过:hover和:focus伪类指定）会在用户把鼠标放置于按钮上方或用Tab键让它获得焦点时提供视觉反馈。创建一个custom-select-open类，通过JavaScript给按钮应用下列样式，使它在菜单展开时看上去像是被按下了，如图17-5所示。

```
.custom-select:hover,
.custom-select:focus,
.custom-select-open {
  background-position:right center;
  border-color:#999;
}
.custom-select:hover .custom-select-button-icon,
.custom-select:focus .custom-select-button-icon,
.custom-select-open .custom-select-button-icon {
  background-position:-500px center;
}
```



图17-5 悬停/获得焦点状态对应的自定义下拉菜单按钮

用绝对定位的方式，让包含可听见角色文字（“select”，即下拉菜单）的span对视力正常的用户隐藏起来，这样屏幕阅读器就仍然能发现它并朗读出来：

```
.custom-select-roletext {
  position: absolute;
  left: -99999px;
}
```

17

对于选项列表，我们会添加一个边框、一张背景图像和一个很大的z-index值，这样下拉菜单就会显示在网页其他元素的上方。我们还会应用overflow属性来有条件地显示垂直滚动条，增强脚本会用这个属性通过程序限制菜单的高度：

```
.custom-select-menu {
  border: 1px solid #b3b3b3;
  background: #e9e9e9;
  z-index: 999999;
  position: absolute;
  margin: 0;
  padding: 0;
  font-size: 1.3em;
  -moz-border-radius: 7px;
  -webkit-border-radius: 7px;
  border-radius: 7px;
  width: 180px;
  cursor: pointer;
  text-shadow: 1px 1px 0 #fafafa;
  overflow: auto;
  overflow-x: hidden;
}
```

我们会移除列表项的内边距和外边距，转而把这些属性应用到选项链接上：

```
.custom-select-menu li {
  padding: 0;
  margin: 0;
  list-style: none;
  clear: both;
}
```

每个列表项的链接都会添加样式，使它看起来像个菜单选项，而非标准的链接，因此给它设置块级属性、内边距和灰色字体颜色，并移除链接默认的文本下划线：

```
.custom-select-menu li a {
  text-decoration:none;
  color:#555;
  display:block;
  cursor:pointer;
  padding:.5em 5px;
  text-shadow:1px 1px 0 #f9f9f9;
}
```

我们想要让按钮显示出恰当的交互反馈。首先，确保不去覆盖浏览器原生的获得焦点状态，也就是在用户通过键盘导航到它上面时，默认应用到获得焦点状态上的点状或发光轮廓线。另外，脚本会在用户悬停或某个选项获得焦点时动态指派一个hover-focus类，从而应用一张稍有不同的背景图像：

```
.custom-select-menu li.hover-focus {
  background:#e6e6e6 url(../images/button-silver.gif) right center repeat-x;
  color:#444;
}
```

用户选择某个选项后，脚本会指派selected类来提供视觉反馈：

```
.custom-select-menu li.selected {
  background:#66b81e url(../images/menu-green.gif) left center repeat-x;
  color:#fff;
}
.custom-select-menu li.selected a {
  color:#fff;
  text-shadow:1px 1px 0 #2d7406;
}
```

给菜单按钮和各个选项链接添加图标来标明礼物包装的类型。给按钮文本的span（custom-select-status）和各个选项链接指派一张单独的组合背景图像，设置内边距让这些容器里的文字不会遮盖图像，并调整各个类里背景图像的位置以显示出对应的图标：

```
#giftwrap-button .custom-select-status,
#giftwrap-menu li a {
  background-image:url(../images/select-icons.gif);
  background-repeat:no-repeat;
  padding-left:40px;
}
.purple-bubbles .custom-select-status,
li.purple-bubbles a {
 background-position:5px 3px;
}
.yellow-swirl .custom-select-status,
li.yellow-swirl a {
 background-position:5px -47px;
}
.blue-snow .custom-select-status,
li.blue-snow a {
 background-position:5px -97px;
}
.pink-sparks .custom-select-status,
li.pink-sparks a {
 background-position:5px -147px;
}
.green-blobs .custom-select-status,
li.green-blobs a {
 background-position:5px -197px;
}
```

应用样式之后，自定义下拉菜单现在如图17-6所示。

我们差不多已经完成了增强标记和样式。在结束之前，需要处理基础标记里的原生select元素及其label。

自定义下拉菜单会完全取代基础select的功能，因此没有理由让原来的select继续在页面上保持可见。用JavaScript给select元素添加select-hidden类，同时也添加一个aria-hidden="true"属性，让select对启用ARIA的屏幕阅读器隐藏：

```
<select name="giftwrap" id="giftwrap" class="select-hidden" aria-hidden=
"true">
 ...
</select>
```

17

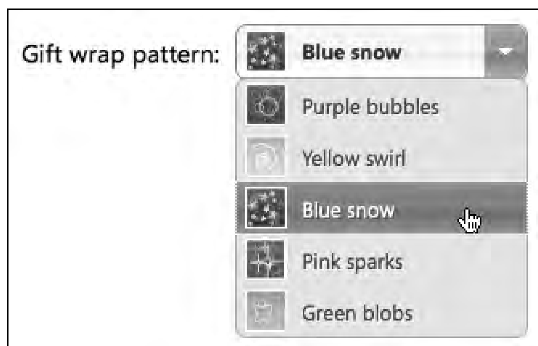


图17-6　各项都带有图标的自定义下拉菜单

为了隐藏select，应用display: none属性：

```
.select-hidden {
  display: none;
}
```

最后，关联基础标签元素（“Gift wrap pattern:”，意为礼物包装图案）和自定义下拉菜单的按钮，方法是更新它的for属性，指向按钮的id（稍后用JavaScript给标签显式设置一个click事件来强化这些元素之间的关系）：

```
<label for="giftwrap-button">Gift wrap pattern:</label>
```

对增强标记和样式的规划至此告一段落。下面创建脚本来生成这些标记，并让它能够工作。

17.2.3　自定义下拉菜单增强脚本

某个浏览器通过能力测试后，我们会用JavaScript把基础标记里的select元素转变成自定义的下拉菜单组件，指派用于开启和关闭菜单的事件，添加键盘支持，并应用ARIA属性使一切都具备可访问性。我们还会在原生select和它对应的自定义组件之间创建一个代理连接，这样表单无论在哪种体验里都能一致地进行提交。

1. 生成增强标记

脚本的第一步是生成增强的自定义下拉菜单标记，并把它插入网页。首先，创建若干变量来存放对原生select元素及其选项属性的引用（本章后面会用它们来构建菜单按钮和自定义选项列表，以及设置当前的选中项）：

```
//对原生select元素的引用
var selectElement = $('#giftwrap');

//获取select的ID
var selectElementID = selectElement.attr('id');

//获取初始选中序号
var initialSelectedIndex = selectElement[0].selectedIndex;
```

```
//获取选中项的文本值
var selectedOptionText = selectElement.find('option')
            .eq(initialSelectedIndex).text();

//获取选中项的类属性值
var selectedOptionClass = selectElement.find('option')
            .eq(initialSelectedIndex).attr('class');

//得到由所有select选项的类所构成的数组，用于快速移除
var allOptionClasses = selectElement.find('option')
      //将选项的类转换成数组
      .map(function(){ return $(this).attr('class'); })
      //把数组合并成类的字符串，类和类之间由一个空格分隔
      .get().join(' ');
```

接下来，根据原生select的ID创建自定义下拉菜单按钮和菜单的ID，然后构建菜单按钮的标记，并将它插入到网页里的原生select之后：

```
//创建按钮和菜单的ID
var buttonID = selectElementID + '-button';
var menuID = selectElementID + '-menu';

//创建空白的菜单按钮
var button = $('<a class="custom-select" id="'+ buttonID
➥+'" role="button" href="#" aria-haspopup="true" aria-owns="'
➥+ menuId +'"></a>');

//创建按钮文本、图标和角色文本的span，并将它放到按钮上
var selectmenuStatus = $('<span class="custom-select-status">'
➥+ selectedOptionText +'</span>')
  .appendTo(button);

var selectmenuIcon = $('<span class="custom-select-button-icon"></span>')
  .appendTo(button);

var roleText = $('<span class="custom-select-roletext"> select</span>')
  .appendTo(button);

//如果指定了select的tabindex属性，就转移过来
if(selectElement.is('[tabindex]')){
  button.attr('tabindex', selectElement.attr('tabindex'));
}

//添加前面定义的选中项的类
button.addClass(selectedOptionClass);

//在select之后插入按钮
button.insertAfter(selectElement);
```


最后，关联自定义下拉菜单的按钮和基础标记里的label元素，并给这个标签应用一个click事件，这样用户点击时它就会把焦点转到按钮上：

```
//关联select的标签到新按钮上
$('label[for='+selectElementID+']')
  .attr('for', buttonID)
```

```
  .bind('click', function(){
      button.focus();
      return false;
});
```

现在我们就准备好处理菜单了。构建选项列表的方法是遍历原生select里的所有option元素，记录各个option的序号（它在列表顺序里的位置）和class属性。然后把完整的选项列表加到body上：

```
//创建菜单ul
var menu = $('<ul class="custom-select-menu" id="'+ menuID+'" role="listbox" aria-hidden="true"
aria-labelledby="' + buttonID+'"></ul>');

//找到selectElement里的所有选项元素
selectElement.find('option')
  //遍历各个选项元素，追踪它们的序号
  .each(function(index){
      //用选项的文本和class属性创建li
      var li = $('<li class="'+ $(this).attr('class') +'"> <a href="#" tabindex="-1" role="option"
      aria-selected="false">' $(this).text() +'</a></li>');
  //如果选项的序号匹配select元素的选中序号
  if(index == initialSelectedIndex){
      //添加选中的类和ARIA
      li.addClass('selected').attr('aria-selected',true);
 }
 //给菜单追加li
 li.appendTo(menu);
});

//追加菜单到页尾，目前还不要隐藏它
menu.appendTo('body');
```

为保险起见，检查菜单的高度并将它限制为300像素。我们在CSS里设置了overflow属性，这样菜单如果超出了最大高度就会显示垂直滚动条：

```
//限制菜单的高度
if(menu.outerHeight() > 300){
  menu.height(300);
}
```

检查完菜单的高度之后，用custom-select-menu-hidden类来隐藏它。用户与菜单按钮交互时，我们会切换按钮上的这个类来隐藏或显示它：

```
//隐藏菜单
menu.addClass('custom-select-menu-hidden');
```

最后，给body元素添加名为application的ARIA role，告知屏幕阅读器这张网页现在包含了交互性组件：

```
//给body添加application角色
$('body').attr('role','application');
```

2. 应用自定义下拉菜单行为

我们会创建自定义的jQuery事件来定义所有的逻辑和行为，它们用于开启和关闭选项列表，

更新某个选项被选中时的菜单按钮反馈，以及更新原生select的值来使它保持同步。这里的每一条都会在脚本里定义一次，然后根据用户与鼠标或键盘的交互触发。

自定义的show事件会移除菜单的custom-select-menu-hidden类，设置它的aria-hidden属性为false，并将它动态定位到按钮的正下方。它还会把焦点放到菜单的当前选中项上，并给按钮添加一个custom-select-open类：

```
//自定义的show事件
menu.bind('show', function(){
  $(this)
      //移除隐藏类
      .removeClass('custom-select-menu-hidden')

      //移除ARIA隐藏属性
      .attr('aria-hidden', false)

      //把菜单放在按钮下方
      .css({
          top: button.offset().top + button.height(),
          left: button.offset().left
      })
      //把焦点转移到选中项上
      .find('.selected a')[0].focus();

      //给按钮添加开启类
      button.addClass('custom-select-open');
});
```

hide事件与show事件里的操作刚好相反，它会移除按钮上的custom-select-open类，并给菜单加回原来的custom-select-menu-hidden类和aria-hidden="true"属性：

```
//自定义的hide事件
menu.bind('hide', function(){
  //移除按钮上的开启类
  button.removeClass('custom-select-open');

  $(this)
      //移除隐藏类
      .addClass('custom-select-menu-hidden')
      //移除ARIA隐藏属性
      .attr('aria-hidden', true);
});
```

toggle事件会有条件地显示和隐藏菜单。如果菜单是隐藏的，那么触发show事件；如果菜单已经是可见的，那么触发hide事件：

```
//给按钮应用mousedown事件
menu.bind('toggle', function(){
  //如果菜单是隐藏的，首先设置它的位置
  if(menu.is(':hidden')){
      //显示菜单
      menu.trigger('show');
  }
```

```
    else{
        //隐藏菜单
        menu.trigger('hide');
    }
});
```

select事件和菜单里的锚元素进行绑定，它会设置当前的选中项，更新菜单按钮的文本和图标，并把选中项代理传回原生的select，从而使表单能正确提交数据：

```
//此事件根据当前的选择更新下拉菜单（并代理传回select）
menu.find('a').bind('select',function(){
  //取消选择之前的菜单选项
  menu
      .find('li.selected')
      .removeClass('selected')
      .attr('aria-selected', false);

  //获取新选中li的类属性
  var newListItemClass = $(this).parent().attr('class');

  //更新按钮图标的类以匹配这个li
  button.removeClass(allOptionClasses).addClass( newListItemClass );

  //更新按钮的文本为这个锚元素的内容
  selectmenuStatus.html( $(this).html() );

  //更新这个列表项的选中属性
  $(this)
      .parent()
      .addClass('selected')
      .attr('aria-selected', true);

  //触发自定义的hide事件来隐藏菜单
  menu.trigger('hide');
//用新的选择来更新原生的select
  selectElement[0].selectedIndex = menu.find('a').index(this);
});
```

接下来，创建逻辑代码，根据用户的交互情况触发自定义事件。在mousedown时触发toggle事件来显示和隐藏菜单，并return false以防止按钮获取焦点：

```
//给按钮应用点击事件
button.mousedown(function(){
  menu.trigger('toggle');
  return false;
});
```

注意　我们使用的是mousedown而不是click，这样用户就能按下鼠标键，拖动到菜单上，然后释放鼠标键进行选择。

我们在菜单按钮上禁用click事件（方法是让它返回false），因为不想让浏览器遵循锚元素的原生click行为：

```
//禁用按钮的click事件（用mousedown/up代替）
button.click(function(){
  return false;
});
```

在用户点击页面上的其他位置时隐藏菜单：

```
//给document绑定click事件来隐藏菜单
$(document).click(function(){
  menu.trigger('hide');
});
```

指定完菜单按钮的行为之后，接着处理菜单。首先，给每个菜单选项链接都绑定一个mouseup事件，它会触发我们的select事件，并return false来防止锚元素获得焦点：

```
 //给菜单应用mouseup事件来选择选项
 //这个事件允许用户进行拖放操作
 menu.find('a').mouseup(function(event){
 //在这个锚元素上触发select事件
 $(this).trigger('select');
 //防止后续原生事件的发生
 return false;
});
```

为了提供视觉反馈，给菜单选项链接绑定mouseover和focus事件，从而给它们的上级列表项添加hover-focus类。我们还会绑定一些事件，在用户离开选项时移除这个类。

```
//hover和focus事件
menu.find('a')
  .bind('mouseover focus',function(){
      //移除之前hover-focus过的选项上的类
      menu.find('.hover-focus').removeClass('hover-focus');

      //给这个选项添加类
      $(this).parent().addClass('hover-focus');
  })
  .bind('mouseout blur',function(){
      //从这个选项上移除类
      $(this).parent().removeClass('hover-focus');
  });
```

接下来，会应用键盘事件来匹配原生select元素的行为，包括：

- 用空格键或回车键打开菜单；
- 用上下左右方向键在菜单选项里导航；
- 用空格键或回车键在菜单里进行选择。

我们会给菜单按钮绑定一个keydown事件来做不少事情。当它被触发时，这个事件会保存对选中项的引用，然后根据按键的keyCode用一条switch语句来应用逻辑代码。

按下空格键或回车键会触发toggle事件来显示或隐藏菜单。按下方向键会让上一个或下一个选项变成选中项，并使它显示在按钮里（不必打开菜单）：

```
//如果在菜单里按下回车键或空格键，就触发mousedown事件
button.keydown(function(event){
```

```
    //找到菜单里的选中列表项
    var currentSelectedLi = menu.find('li')
            .eq( selectElement[0].selectedIndex );

    //处理不同的按键事件
    switch(event.keyCode){
        //如果在菜单里按下回车键或空格键，就触发toggle事件
        case 13:
        case 32:
            button.trigger('toggle');
            return false;
        break;

        //向上或向左方向键
        case 37:
        case 38:
            //如果之前存在选项，就选中它
            if( currentSelectedLi.prev().length ){
                currentSelectedLi.prev().find('a')
                    .trigger('select');
            }
            //防止原生滚动
            return false;
        break;

        //向下或向右方向键
        case 39:
        case 40:
            //如果之后存在选项，就选中它
           if( currentSelectedLi.next().length ){
               currentSelectedLi.next().find('a')
                   .trigger('select');
           }
           //防止原生滚动
           return false;
       break;
    }
  });
```

我们会给菜单应用一个逻辑相似的keydown事件，不同之处在于方向键在移动焦点的同时不会改变选中项。这样用户就可以用方向键浏览各个选项，然后用Tab键在不进行选择的情况下离开。按下空格键或回车键会触发获得焦点的锚元素的`select`事件，关闭菜单，然后将焦点转回到菜单按钮上。所有这些按键操作都会返回`false`。

按下Tab键时，隐藏菜单并将焦点转回到菜单按钮上，这样焦点就会跳到下一个可获得焦点的网页元素上，这么做是为了确保制表键焦点遵循用户期望的制表键顺序：

```
  //菜单的keydown事件
  menu.keydown(function(event){
    //switch根据按键情况应用不同的逻辑
    switch(event.keyCode){
        //如果在菜单里按下的是回车键或空格键，就触发mouseup事件
        case 13:
```

```
        case 32:
            // 触发select事件
            $(event.target).trigger('select');
            button[0].focus();
            return false;
        break;

        //向上或向左方向键
        case 37:
        case 38:
             //如果之前有可选项，就让它获得焦点
             if( $(event.target).parent().prev().length ){
                 $(event.target).parent().prev().find('a')[0]
                     .focus();
             }
             //防止原生滚动
             return false;
         break;
         //向下或向右方向键
         case 39:
         case 40:
             //如果之后有可选项，就让它获得焦点
             if( $(event.target).parent().next().length ){
                 $(event.target).parent().next().find('a')[0]
                     .focus();
             }
             //防止原生滚动
             return false;
         break;
         //Tab键使焦点返回到菜单按钮上
        //让浏览器能够自动把焦点转移到下一个可获得焦点的网页元素上
         case 9:
             menu.trigger('hide');
             button[0].focus();
         break;
   }
});
```

最后，因为原生和自定义的下拉菜单执行相同的功能，所以我们会应用一些属性来隐藏原生的下拉菜单：

```
selectElement.addClass('select-hidden').attr('aria-hidden', true);
```

17.3　让自定义下拉菜单更进一步：给选项添加高级样式

上面的自定义下拉菜单范例表现了简单的文本格式和图像。有些设计可能会借助更丰富的文字格式来应用视觉层级。举个例子，请考虑这个要求用户选择相册的自定义下拉菜单，如图17-7所示。

我们无法用原生的select元素实现这种级别的视觉样式，因为它的能力有限（它的子option元素不能包含HTML标签），所以不得不创建一个自定义的下拉菜单来实现它。重要的问题在于，

有没有办法能用原生的select元素来创建这个增强自定义下拉菜单（以保留它的完整可访问性），然后用上面范例里的数据挖掘方式创建一个通用脚本来转变它？



图17-7　带有多行文本格式样式的自定义下拉菜单设计

乍一看，用原生select里的内容来表达这种级别的内容层级似乎是不可能的，但其实这是可以做到的。我们只需格式化各个原生option里的文本来划分出独立的三行（标题、描述和日期），让它们为自定义下拉菜单里的样式做好准备。然后就可以在自定义下拉菜单里用JavaScript解析这些文本并格式化成HTML。

首先，考虑如何能在不牺牲基本体验可用性的前提下，用HTML的字符和标点符号来制作增强版里的格式。举个例子，可以用连字符和括号划分各行文字：

```
My Title -- This is a description (Created January 12, 2009)
```

我们会把这种文本格式放入select元素的基础标记里，如图17-8所示。

```
<select name="album" id="album">
  <option value="albumA">Summer Party -- Fun at the lake house (Created August 12, 2009)</option>
  <option value="albumB">Kate's Birthday -- Cake and presents (Created May 13, 2009)</option>
  <option value="albumC">Dance Recital -- First one of the year (Created February 6, 2009)</option>
</select>
```

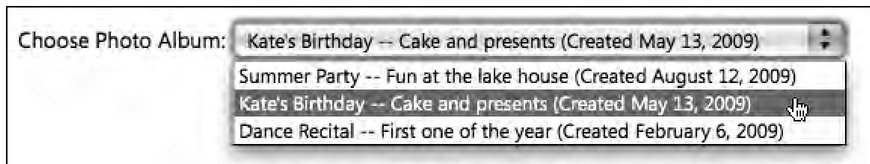


图17-8　基本体验里用简单标点符号格式化后的文本

然后，为了转换列表项标记，用一个JavaScript正则表达式来寻找并替换随HTML添加的标点符号。正则表达式提供了许多由特殊字符构成的模式来寻找文本字符串，而JavaScript让我们可以对它们进行操作。我们会编写一个正则表达式来寻找连字符和括号，并把它保存在一个名为pattern的变量里：

```
var pattern = /([\s\S]+)\-\- ([\s\S]+) (\(([\s\S]+\))/;
```

这个表达式会截取我们想要匹配的3个片段：标题、描述和日期（上面代码里的高亮部分）。

JavaScript会自动给每个片段都指派一个变量名，所以可以很方便地用变量$1、$2和$3来指代正则表达式里对应的各个部分。用这些美元符号变量所引用的3个正则表达式片段来构建各个文本行的新增强标记，并将它保存在一个名为replacement的变量里：

```
var replacement = '<span class="option-title">$1</span>'+
  '<span class="option-description">$2</span>'+
  '<span class="option-date">$3</span>';
```

增强脚本生成自定义下拉菜单后，我们会运行一小段脚本来获取增强标记里的各个选项文本，然后将它们替换成格式化后的对应标记，方法是在JavaScript的replace方法里引用pattern和replacement变量：

```
menu.find('li a').each(function(){
  //获取锚元素的文本
  var text = $(this).text();

  //把锚元素的HTML替换成新格式化后的标记
  $(this).html( text.replace(pattern,replacement) );
});
```

生成的增强标记现在就包含了span标签，它们替代了我们在基础标记里使用的标点符号：

```
<ul id="album-menu" class="custom-select">
  <li class="option-selected">
      <span class="option-title">Summer Party</span>
      <span class="option-description">Fun at the lake house</span>
      <span class="option-date">Created August 12, 2009</span>
  </li>
  ...
</ul>
```

把标记插入网页之后，就可以用CSS来给这些span元素添加样式，使它们看上去接近我们的目标设计：

```
#album-menu li a span.option-title {
  font-weight: bold;
  display:block;
}
#album-menu li a span.option-date {
  display:block;
  font-size: .9em;
  font-style: italic;
}
```


17.4 使用自定义下拉菜单脚本

本书附带的演示和代码（位于www.filamentgroup.com/dwpe）包含一个名为jQuery.selectmenu.js的脚本，它能用本章概述的原则自动创建自定义下拉菜单，并提供完整的ARIA和键盘导航支持。

这个脚本还模拟了原生select的搜索输入提示行为：当用户在获得焦点的自定义下拉菜单里按下字母键时，它会选中第一个以这个字母开头的选项。

要在你的网页里使用这个脚本，请下载并引用下拉菜单页里显示的那些文件，然后在网页里对某个select元素调用selectmenu方法，就像下面的代码所演示的：

```
$('select#foo').selectmenu();
```

maxHeight选项用来设置选项列表菜单的最大像素高度。如果选项列表的高度超过了这个值，就会出现一个垂直滚动条：

```
$('select#foo').selectmenu({
  maxHeight: 300
});
```

要格式化自定义下拉菜单里各个菜单项的文本，可以定义一个接受单个参数（text）的函数，这个参数代表了各个列表项的文本。举个例子，这段代码会用一个span围住自定义下拉菜单里的各个选项：

```
$('select').selectmenu({
 format: function(text){
        //将自定义选项文本包装在span中
        return ''+text+'</span>';
  },
});
```

这个格式选项可用于创建17.3节讨论的多行分别格式化的例子。

在替换任何原生表单元素时，我们总是想确保增强的组件至少有着与原生元素相同的能力，然后才会用非原生的功能来扩展这个组件。这就意味着它的所有功能（比如键盘快捷方式和可访问性属性）必须和原生控件一样表现良好，才能确保那些期望自定义下拉菜单能比拟标准select控件的用户不会感到受挫或者困惑。

我们通过本章的自定义下拉菜单组件寻找到了一种方法，既能获取原生select的所有优点，又能叠加增强信息以丰富用户体验，还采用了通用的逻辑，减轻了开发者的负担。

第 18 章

列表生成器

随着Web应用程序逐渐接近桌面应用程序的功能，我们会遇到越来越多某个UI元素没有原生HTML先例的情况。列表生成器组件（list builder）就是这样一个例子。

列表生成器最简单的形式是在一个可编辑的区域内归组若干单词或短语。各个条目通常会装在带有边框或背景色的“盒子”里，使列表便于浏览、编辑和删除。而且因为每个条目都被视为一个独立的单元，所以可以给它们添加更为丰富的交互手段，比如多选、拖放、操作按钮和上下文菜单。

列表生成器这种交互方式可以在各种常见的Web应用程序里找到，包括：

- 在带有标签系统的网站里，用户输入若干单词或短语来描述照片、文档、地图兴趣点、视频或书签；
- 某些Email应用程序会帮助创建To（收件人）、Cc（抄送）和Bcc（秘密抄送）字段里的收件人列表；
- 在日用百货送货服务和其他基于列表的购物网站里，用户可以快速添加、移动和删除列表项。

许多复杂的列表生成器加入了自动完成功能，会给出系统里现有的建议值，例如已经用于其他物品的标签，或者用户地址簿里保存的电子邮件信息，如图18-1所示。

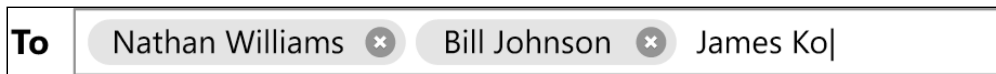


图18-1　一个消息接收者列表生成器组件

本章会描述如何使用可访问的标准HTML表单控件构建一个简单的列表生成器，然后用JavaScript把它转变成以基本版功能为基础的交互式组件，同时还保留原生元素所有的可访问性功能。然后，讨论如何增强一个标准的列表生成器，给它添加多项选择、拖放排序、自动完成和上下文菜单等更为复杂的功能。

18.1　X 光透视

首先，我们会考虑一个相当简单的列表生成器，它的作用是给某张照片添加描述性的标签，如图18-2所示。



图18-2　照片标签列表生成器的目标设计

这个目标设计展示了归组到可编辑文本框中的一列标签，其中每一项都带有边框和背景图像，使它看起来像个图标式标签。每个标签都附带一个小小的移除按钮来快速一键删除此标签，用户还可以方便地在文本框容器中输入，从而添加新标签。

这个列表生成器的设计包含了许多高级功能：自定义样式，各个条目上的移除按钮，以及一个自定义样式的Save Tags（保存标签）提交按钮。但是用X光透视这个设计时，我们发现上述功能不属于这个工具的核心部分：列表生成器的关键功能是提供一种途径来输入和提交一个或多个单词和短语。我们的基础标记可以用一个文本输入表单元素来实现列表生成器所有的关键功能，它接受一个由空格或逗号分隔的列表，后面跟着一个标准的提交按钮。

有两个HTML元素（文本`input`和`textarea`）可以满足这些要求。它们都具备可能有用的独特功能。

- 文本`input`内建了限制条件：它接受单行文本，而且只显示它的固定尺寸能够容纳的字符数量。（虽然`input`元素从技术上说可以接受超出其尺寸大小的内容，但是用户需要用方向键才能看到超出可见区域的字符。）它对那些限制字符数量的案例来说是理想的选择。
- `textarea`容许更大的灵活性：它可以接受多行输入，而且会在内容超出可见区域时自动添加滚动条。

有经验的Web用户通常会根据显示的元素类型来理解某一系统的限制，并假定字段的尺寸暗示了可接受答案的长度。在这个照片标签范例里，我们不想限制用户给任意照片添加的标签数量，因此在基础标记里使用一个`textarea`元素来帮助用户了解这一点，如图18-3所示。



图18-3　列表生成器的基本体验

增强版列表生成器组件所包含的功能要比textarea原生支持的更为复杂（举个例子，每个照片标签都有自定义的样式和一个内建的移除按钮），所以增强版组件应该由通过JavaScript插入网页的新标记构建。我们会给一个标准的无序列表（ul）添加样式来容纳标签，它的最后一项总是包含一个空白的文本input以便进行新的内联输入。

为了简化服务器端的表单处理工作，我们会遵循标准代理模式：增强版的列表生成器组件会在页面加载时创建，随后脚本会检查基础标记里的textarea，为每个现有值生成一个无序列表项。添加或移除生成器组件里的某个标签时，textarea会被更新，使两者无论何时都保持同步。在屏幕上，我们会对用户隐藏基础HTML元素，但让它留在网页的标记里，这样无论是基本版还是增强版的网页都能以相同的方式提交表单。

为了保持可访问性，增强版列表生成器的构建方式必须能让屏幕阅读器用户理解这个组件是什么，以及如何使用它。我们会用ARIA属性充分描述它的功能，同时让辅助技术能够理解。

18.2　创建列表生成器

首先用一个标准的textarea来标记网页，它可以用由逗号分隔的一组简单的值填充。然后把这些值转换成增强体验里的一组照片标签，这些标签带有样式且可编辑。

18.2.1　基础标记和样式

列表生成器的基础标记很简单：它由一个textarea和一个描述其用途的配套label组成。label元素的for属性匹配这个textarea的id属性，这样就能正确地为屏幕阅读器关联它们。在用户输入标签之前，这个textarea不包含任何值：

```
<label for="tags">Photo tags:</label>
<textarea id="tags" name="tags"></textarea>
```

如果用户之前曾经给某张照片输入过标签，那么网页加载时textarea元素会被预先填充一个值，这个值是一列由逗号分隔的现有照片标签。在这个例子里，假设用户已经给照片指派并保存了两个标签（yellow和flowers），它们在网页加载时会一起出现：

```
<textarea id="tags" name="tags">yellow, flowers</textarea>
```

这段标记就位后，可以给列表生成器应用一些非常简单的CSS规则，它们对几乎所有浏览器而言都是安全的。

- 一个全局的font-family列会指派给body标签和textarea元素（这一点特别重要），因为文本输入的表单元素在大多数浏览器里都不会自动继承字体样式。
- 一个块级的display属性会设置在label上，使它位于在textarea上方（否则它会位于左侧）。

在基本样式表里，我们的安全样式看起来如下所示：

```
body, textarea { font-family: "Segoe UI", Arial, sans-serif; }
label { display: block; }
```

瞧！列表生成器的基础标记就构建完了，如图18-4所示。

Photo tags:
yellow, flowers, nat|
Save Tags

图18-4　应用安全样式后的照片标签生成器基础标记

我们不需要给基本体验添加任何可访问性功能，因为所有的原生HTML表单元素都是可访问的。举个例子，列表生成器的所有常规功能都可以用键盘命令访问和使用：用户可以按下Tab键使textarea获得焦点，输入由逗号分隔的一列值，然后按下Tab使“保存标签”按钮获得焦点，从而提交表单。

18.2.2　增强标记和样式

列表生成器组件的增强标记会在能力测试通过后由JavaScript插入到网页里，并把它的内容值传回基础textarea，用于表单提交。

首先，textarea的功能会被列表生成器组件所取代，因此我们会让它对所有用户隐藏，包括屏幕阅读器（我们想让所有人都操作列表生成器组件，而完全忽略textarea）：

```
textarea#tags { display: none; }
```

注意　请记住这个textarea只是对用户隐藏。构建列表生成器脚本时，把基础textarea设置成一个代理，用于提交在列表生成器里输入的那些值。

现在万事俱备，可以构建增强版列表生成器了。用一个class为listbuilder的无序列表把标签归组成语义的形式，使它们能够应用盒子风格的样式：

```
<ul class="listbuilder">
  ...
</ul>
```

ul里所含的每个列表项（li）都代表一个独一无二的照片标签。

这个无序列表应该始终包含一个带有文本输入框的列表项，以供用户输入新的标签。我们会用一个具有描述性的类（listbuilder-entry-add）来标识它。输入框的title属性会在鼠标悬停时显示出一条工具提示，并为屏幕阅读器提供朗读指导：

```
<li class="listbuilder-entry-add">
  <input type="text" value="" title="To add an item to this list, type a
  name and press enter or comma." />
</li>
```

注意 我们有意省略了这个input上的name属性，因为纯粹只是用它来收集和临时存放某个新输入的标签，并不需要将它的值和其余表单数据一起提交。

列表生成器脚本会在网页载入时为textarea里的每一个条目都生成一个新标签。各个标签的标记都是一个列表项，里面用一个span围住照片标签的文本。

```
<li class="listbuilder-entry">
  <span class="listbuilder-entry-text">yellow</span>
</li>
```

每当用户输入一个单词或短语，然后加上逗号或者直接按回车键，脚本就会用input的值创建一个新的列表项并将它插入到标签列表的末尾，可编辑列表项的前面。

每个照片标签还带有一个属于listbuilder-entry-remove类的移除链接，它的文本值是Remove（移除）。我们会用CSS隐藏这个链接的文本，并用一个紧凑的“x”背景图标来取代它，另外再给它添加一个title属性，用文字解释这个按钮的功能（这个属性会同时用于原生的工具提示和屏幕阅读器的有声反馈），以及一个名为button的ARIA role来描述此链接的用途：

```
<li class="listbuilder-entry">
  <span class="listbuilder-entry-text">yellow</span>
  <a class="listbuilder-entry-remove" title="Remove yellow from the list."
  href="#" role="button">Remove</a>
</li>
```

最后，处理基础的label元素。在基本体验里，label只描述了textarea；在增强体验里，它则需要描述所有这些新的标记（无序列表结构，用于输入新条目的文本输入框，文本span和移除按钮），在这种情况下单个label元素是不够用的。为了让它的增强版角色更具意义，用JavaScript把它转变成一个HTML标题：

```
<h2>Photo tags:</h2>
```

在这个阶段，我们已经确立了列表生成器的增强标记：它是一张简单的标签无序列表，后面跟着移除链接和一个可编辑的文本输入框，如图18-5所示。

接下来，添加CSS来装饰增强体验里的列表生成器，并给替换label元素的列表生成器标题应用更大的字号和粗细：


```
h2 {
  font-size:1.3em;
```

```
  margin:.5em 0;
  font-weight:bold;
  text-transform:uppercase;
}
```

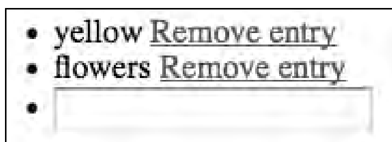


图18-5　用于两个标签和文本输入框的无样式增强标记

我们会给列表生成器的容器（ul）应用一些样式，比如border和overflow属性，来模拟标准textarea的外观和滚动行为。这个列表生成器带有中灰的背景色以突出各个标签，并使用CSS3的border-radius属性在支持这一属性的浏览器里圆化边角：

```
ul.listbuilder {
  padding: 0;
  margin: 0;
  background:#88898a;
  list-style:none;
  border:1px solid #4D4D4D;
  overflow:auto;
  cursor:text;
  -moz-border-radius:5px;
  -webkit-border-radius:5px;
  border-radius:5px;
}
```

当光标位于列表生成器上时它的背景色会转变成白色，使它看起来更像是一个可编辑的活动文本字段，如图18-6所示。这一颜色变化靠的是根据input的焦点状态给列表附加listbuilder-focus类：

```
ul.listbuilder-focus { background: #fff; }
```



图18-6　列表生成器组件的默认（上图）状态和获得焦点（下图）状态

我们会让所有列表项和可编辑输入框向左浮动，使它们并排显示，并添加一点点外边距来隔开它们：

```
ul.listbuilder li {
  float:left;
  font-size:1.3em;
  margin: 2px 0 2px 2px;
  color: #333;
}
```

这个可编辑文本框应该能无缝融入列表生成器的容器里，所以我们会移除它的边框、内边距、外边距和轮廓线，并设置字体大小为1em，使它继承列表的字体大小：

```
ul.listbuilder input {
  font-size:1em;
  margin:0;
  padding:0;
  background:transparent;
  border:0;
  outline:0;
  line-height:1.7;
  color: #fff;
  width: 3em;
}
```

为了让各个标签看起来更像标签，我们会应用两张背景图像：一张在上，指派给li，另一张在下，指派给span。以这种方式使用这两张“滑动门”效果的背景图像，可以确保背景能灵活适应文字大小，比如用户用浏览器控件调整文字大小时。把span设置成display: block，使它填满可用空间，并在右边额外添加25像素的内边距来容纳移除链接：

```
ul.listbuilder li.listbuilder-entry {
  position:relative;
  cursor:default;
  border-right:1px solid #7d7d7d;
  background:#eee url(../images/bg-tag-top.png) top left no-repeat;
}
ul.listbuilder span.listbuilder-entry-text {
  display:block;
  padding:.2em 25px .4em 18px;
  background:url(../images/bg-tag-bottom.png) bottom left no-repeat;
}
```

最后，用一个很大的text-indent负值来隐藏移除链接的文本，然后用背景图标取代它。我们还会绝对定位这个链接，这样它就会始终位于标签文本的右侧：

```
ul.listbuilder li.listbuilder-entry a.listbuilder-entry-remove {
  position:absolute;
  right:.3em;
  top:50%;
  margin-top:-5px;
  width:11px;
  height:11px;
  text-indent:-99999px;
  overflow:hidden;
  background:url(../images/icon_remove.png) 0 0 no-repeat;
}
```

18.2.3 列表生成器脚本

能力测试在运行时会插入增强CSS和脚本，在通过测试的浏览器里创建列表生成器组件。列表生成器脚本会做许多事。

(1) 生成增强版列表生成器的标记，用textarea里找到的初始值填充列表。

(2) 根据用户的输入添加新标签，实现方式是等待文本输入框里的键盘事件。

(3) 创建一个代理函数，用列表生成器里的标签组持续更新原有的textarea，使它们始终保持同步，便于提交表单。

(4) 给列表应用鼠标点击行为来移除标签，显示视觉反馈，以及让光标聚焦到可编辑文本输入框里。

(5) 把基础标记里的label元素转变成语义更加恰当的增强控件标题。

注意　本章描述的脚本编写原则无论对textarea还是input都适用，因为两者的增强体验是一样的。

---

1. 生成增强标记

列表生成器脚本会根据基础标记里某一个textarea的特性生成组件。首先，确定哪个textarea将成为列表生成器，然后收集它的相关信息，包括它的宽度和当前值（考虑到它可能预先填充了一组标签）。和其他脚本范例一样，我们会对这些变量使用jQuery表示法。

```
var textarea = $('textarea#tags');
var textareaWidth = textarea.width();
var textareaValue = textarea.val();
```

用这些变量创建一个空白的ul作为列表生成器的容器，并将它的宽度设置成原有textarea的宽度：

```
var listbuilder = $('<ul class="listbuilder"></ul>')
          .width( textareaWidth );
```

生成列表生成器组件内容的第一步是为textarea里的现有值创建标签。利用保存在textareaValue变量里的textarea值，为每个用逗号或换行符分隔的单词或短语都生成一个新的li，然后把它们作为标签附到列表生成器上。

为此，用JavaScript原生的split方法把textarea里的值拆分成一个值数组，做法是指定用于划分各个条目的字符。举个例子，要在每个逗号的位置拆分值，传递一个逗号字符串作为split方法的参数：

```
textareaValue.split(',');
```

split方法还接受正则表达式以匹配更复杂的模式。为了在恰当的位置拆分这个用于列表生成器的字符串值，使用正则表达式/[,\n]/，它会寻找每一处逗号（,）或换行符（\n）。（这里的斜杠表示正则表达式的开头和结尾，方括号里描述了一个或多个用于匹配的字符。）我们会创建一个新的变量（textareaValueArray）来存放拆分后的值数组：

```
var textareaValueArray = textareaValue.split(/[,\n]/);
```

使用jQuery的each方法遍历这个数组，给每个标签值都生成一个列表项，然后把它们一次性附加到ul上以优化性能。如果这个数组没有任何值，each方法就只会做0次循环，不会产生错误：

```
$.each( textareaValueArray, function(){
  /*创建对当前项目值的引用 */
  var val = this;

  /* 给列表生成器增加一个新的列表项 */
  listbuilder.append('<li class="listbuilder-entry">
  ➥<span class="listbuilder-entry-text">'+val+'</span>
  ➥<a href="#" class="listbuilder-entry-remove"
  ➥title="Remove '+val+' from the list." role="button">
  ➥Remove</a></li>');
};
```

接下来，我们会附上可编辑的列表项，让用户给列表添加新的标签。如果没有已存在的标签，这个输入框就会是列表生成器容器里的唯一一项；如果有其他项，它将始终是列表里的最后一项。

```
listbuilder.append('<li class="listbuilder-editable">
➥<input type="text" value=""
➥title="Type a name and press enter or comma
➥to add it to the list." /></li>');
```

最后，把列表生成器插入到网页里：

```
listbuilder.insertAfter( textarea );
```

2. 让原本的textarea保持同步

虽然原本的基础textarea被隐藏到视野之外，但它在后台仍然是必要的，用于获取标签值并在用户点击“保存标签”按钮时提交给服务器。如此使用隐藏textarea之后，就能在基本体验和增强体验里用同一种方法处理表单。

为此，使用一种“代理”模式：当用户与增强版列表生成器进行交互时，我们的脚本会用当前的这组值更新原本的隐藏textarea。脚本会遍历列表里的所有条目，把它们的值保存在由逗号分隔的字符串里，然后用这个字符串设置textarea的值。我们会把这段逻辑写入一个独立的函数（updateValue），这样每当用户通过添加、编辑或删除标签来更新列表时，我们就可以调用它：

```
function updateValue(){
  /* 创建一个数组来存放新值 */
  newTextareaValue = [];

  /*遍历各个列表项的文本，把它添加到数组中 */
  listbuilder.find('span.listbuilder-entry-text').each(function(){
      newTextareaValue.push( $(this).text() );
  });

  /* 添加input元素的值作为最后一个列表项 */
  newTextareaValue.push( listbuilder.find('input').val() );

  /* 把这些列表项用逗号连接起来，然后填入textarea */
  textarea.val( newTextareaValue.join(',') );
};
```


3. 给列表应用点击行为

从列表里移除条目的脚本非常简单：用户点击移除按钮后，它对应的列表项会从页面中移除，

textarea的值则会通过调用updateValue函数更新为新的列表。给列表生成器的容器一次性添加这个点击事件，然后用事件指派确保所有的移除链接（包括列表里现有的以及后续可能会添加到列表的那些）都会获得这个行为。

```
listbuilder.click(function(ev){
  //创建对被点击元素的引用
  var clickedElement = $(ev.target);

  //检查被点击元素是否是一个移除链接
  if(clickedElement.is('a.listbuilder-entry-remove') ){

      //移除上级列表项
      clickedElement.parent().remove();
  }

  //让点击事件返回false
  return false;
});
```

我们需要给列表生成器的点击事件再添加一种情形：当用户点击列表生成器但不是为了移除标签时，我们想把光标的焦点移到可编辑输入框字段上。这样，列表生成器就会感觉像是一个原生的textarea：

```
listbuilder.click(function(ev){
  var clickedElement = $(ev.target);
  if(clickedElement.is('a.listbuilder-entry-remove') ){
      clickedElement.parent().remove();
  }
  else {
      //把焦点移到最后一项的input元素上
      listbuilder.find('input').focus();
  }
  return false;
});
```

我们还想在列表生成器获得焦点时提供视觉反馈。为此，利用输入框的focus和blur事件来添加或移除之前创建的listbuilder-focus类：

```
listbuilder.find('input')
  .focus(function(){
      listbuilder.addClass('listbuilder-focus');
  })
 .blur(function(){
     listbuilder.removeClass('listbuilder-focus');
 });
```

4. 添加新标签

当用户在获得焦点的文本输入框里按下某个键时，我们会把键值传回基础的textarea让这些控件保持同步。如果用户按下的是用于添加标签的两个键（逗号或回车键）之一，就会在增强版列表生成器控件里生成一个包含input值的标签。用keydown事件给列表生成器的容器绑定所有这些逻辑，这个事件会在按下某个键，但相应的字符（回车或逗号）尚未添加到输入框的值时触发：

```
listbuilder.find('input')
  //添加keydown事件
  .keydown(function(ev){

      //保存对input的引用
      var input = $(this);

      //检查按键是否是逗号（188）或者回车（13），并且值不是空白的，或者仅有一个逗号
      if( (ev.keyCode == 188 || ev.keyCode == 13)
          && input.val() != '' ){

      //保存对逗号之前的值的引用
      var val = input.val().split(',')[0];

      //在它之前插入一个新的列表项，其中包含新的文本
      input.parent().before( '<li class="listbuilder-entry">
      <span class="listbuilder-entry-text">'+val+'</span>
      <a href="#" class="listbuilder-entry-remove"
      title="Remove '+val+' from the list." role="button">
      Remove</a></li>');

      //重置input的值
      input.val('');
  }

  //无论上面这些条件的结果如何
  //如果按下的键是逗号或者回车，就阻止它完成按键事件
  if(ev.keyCode == 188 || ev.keyCode == 13){
      ev.preventDefault();
  }
});
```

触发keyup事件后，脚本会运行这个updateValue函数来确保textarea始终包含了列表生成器里所有最新的条目，包括当前正在输入的不完整条目：

```
listbuilder.find('input')
//指派keyup事件
.keyup(function(){
  //更新textarea
  updateValue();
});
```

最后，用一个标题来替换基础的label元素，用于整个列表生成器控件：

```
//通过for属性找到标签
var label = $('label[for='+ textarea.attr('id') +']');

//使用jQuery的replaceWith方法把标签替换为一个h2元素
label.replaceWith('<h2>'+label.text()+'</h2>');
```

我们的脚本现在已经能把一个普通的textarea转变成交互性丰富的列表生成器，可以通过添加样式来满足任何应用程序的需求，如图18-7所示。

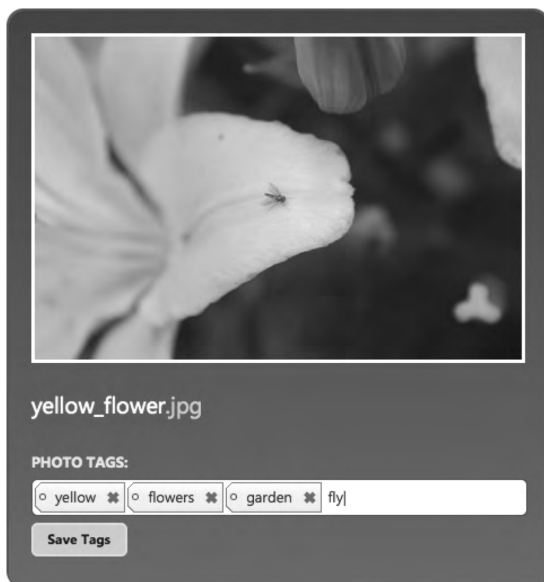


图18-7　最终的增强版列表生成器组件

18.3　让列表生成器更进一步：多项选择、排序、自动完成和上下文菜单

到目前为止，我们分析了从textarea创建列表生成器的基本原理。列表生成器创建出来之后，可以给它添加一系列额外的功能，从而进一步提升这个列表生成器的用处和可用性。

18.3.1　多项选择

同时选择和操作多个项目（方法是按住Control或Shift键然后点击列表生成器里的各个项目），能让人感觉它更像是一个桌面风格的高级组件，而且还消除了不必要的重复操作，从而显著提升了用户的效率。可以添加逻辑代码来实现这个功能，方法是在用户点击某项时留意特定的键盘事件。

举个例子，用户可能想编辑“收件人”字段里的一长串邮箱地址。启用多项选择后，就可以按住Shift键并点击（或者使用方向键）选择多个连续的邮箱地址，也可以按下Control键选择不同列表位置上的多个邮箱地址。

请注意，本书附带的列表生成器插件脚本（位于www.filamentgroup.com/dwpe）会自动包含这一功能。

18.3.2　拖放排序

如果在某些情况下列表生成器里的项顺序很重要，你可以考虑加入点击拖动功能来支持列表

项的重新排序。这一行为可以使用jQuery UI的可排序插件实现（位于jQueryUI.com）。加入jQuery UI的核心和可排序插件，并对列表生成器的容器调用`sortable`方法后，里面的列表项就可通过拖放操作来重排顺序了。

18.3.3 自动完成

许多列表生成器的实现都带有自动完成功能，它会在用户输入新条目时提供一张建议值列表。这个功能用jQuery UI的自动完成组件很容易实现。自动完成功能的配置根据不同的数据来源和结构有所区别，要了解jQuery UI自动完成组件的更多功能信息或者下载这个组件，请访问jQueryUI.com。

实现自动完成功能时，值得考虑的一点是要不要在基本体验里同样提供自动完成建议列表，具体做法是设置一个导航到另一个网页的链接，建议值用单选按钮或复选框的形式放在那个网页里。举个例子，某个电子邮件收件人列表生成器可以在textarea边上放置一个“地址簿”（Address Book）链接，用来将用户导航至另一张带有联系人核对清单的网页。用户可以选择若干联系人的地址，然后点击“添加收件人”（Add Recipients）按钮生成收件人列表。在增强体验里，联系人列表可以用HTML、JSON或XML格式借助Ajax获取，然后填入自动完成菜单以实现更快速和更具交互性的体验。

18.3.4 上下文菜单

另一种扩展列表生成器的有力方式是给每个列表项都添加一张上下文菜单，使用户能快速进行相关的操作。举个例子，在Apple Mail里，收件人、抄送和秘密抄送字段里的列表项都带有一个上下文菜单，可以很方便地复制或保存邮箱地址，从列表中移除地址，或者启动一个与此人的新会话或给此人发送邮件。

这项功能没有包括在我们的插件里，但是有许多jQuery和非jQuery的插件提供了各种方法，给每个列表项都绑定一个菜单事件行为。

18.4 使用列表生成器脚本

本书附带的列表生成器演示和代码（www.filamentgroup.com/dwpe）包含一个脚本：jQuery.listbuilder.js。它不但具有本章描述的所有基本功能，还具有支持多项选择的键盘行为以及用Backspace键移除项。下面介绍如何使用这个脚本。

(1) 下载并引用jQuery库（1.3.1或之后的版本）和本书网站里包含的脚本（jQuery.listbuilder.js），以及在线演示里用到的那些CSS文件与图像。

(2) 将插件指向某个特定的`textarea`或文本`input`，并调用`listbuilder`方法：

```
$('textarea').listbuilder();
```

(3) 根据个人喜好设置选项。这个插件可以使用下列选项进行配置。

- `delimChar`：用于将textarea内容拆分成独立条目的字符或字符串，比如一个逗号或者换行符。
- `width`：列表生成器容器的宽度，默认是它所取代的textarea的宽度。
- `completeChars`：一个用来创建新项目的键码数组。举个例子，用户输入某个单词或短语后，他们可以按下逗号或回车键来添加一个标签。键码的完整列表可以在Mozilla Developer Center上找到（http://t.cn/zRlQ9if）。
- `userDirections`：作为title属性应用到新input上的指导文字。默认的指导文字是“输入一个名称然后按回车或逗号键把它添加到列表中”（Type a name and press enter or comma to add it to the list）。
- `labelReplacement`：HTML字符串形式的一个新标签或一组嵌套标签，用于替换最初关联到textarea上的label元素，比如`<h2 class="list-builder-heading"></h2>`。脚本会抓取label的内容并把它插入指定字符串的最内层标签里。这个选项有一个很好的示例，可以在本书附带代码里的列表生成器JavaScript中找到。

(4) 覆盖任意选项的默认值，具体做法是给listbuilder方法传递一个键值对形式的对象：

```
//用破折号而非逗号/换行符来拆分文本域的值
$('textarea').listbuilder( {delimChar: '-'} );
```

这个列表生成器插件生成的元素和类都与本章介绍的元素和类相匹配。可以使用本书网站里代码范例所包含的CSS文件，加上这一章所展示的设计，把它们作为一个起点来构建你的项目。

列表生成器提供的丰富实例展示了利用X光方法，如何把一个复杂的交互组件映射回非常简单的HTML元素，从而确保对所有设备的兼容性。它还阐明了一个事实，即不是每一项增强功能（比如自动完成和上下文菜单）都需要移植到基本体验里才能提供完美可用的交互性。

第 19 章

文件输入控件

随着各个网站纷纷扩展它们的内容分享和协作功能，用户逐渐选择上传文件（照片、视频、文档，甚至安全信息）并通过基于“云”系统的Web应用程序来保存它们，而不是本地储存在他们自己的电脑里。依赖浏览器上传文件的常见应用包括Flickr或YouTube等照片或视频分享网站，Facebook等社交网站，以及谷歌文档等基于Web的生产效率套件。

HTML提供了一个很方便的原生表单控件来访问本地文件：type设置为file的标准input元素。这个文件input显示为一个用于提供反馈的文本字段和一个按钮。这个按钮会启动一个标准的操作系统对话框，供用户浏览他们本地硬盘里的内容。用户选择了某个文件后，所选文件的路径就会填入文本反馈字段。

不过标准的文件input从样式和可用性的角度看不甚完美。无法用CSS给原生的文件input添加样式，而文本字段和按钮的外观与尺寸在各种浏览器和操作系统里可能会千差万别（甚至连按钮用词可能都不一样）如图19-1所示。最重要的是，大多数浏览器渲染出的固定尺寸文本字段通常太小，不足以显示完整的文件路径，于是用户必须在文本字段里滚动才能看到最关键的用户反馈信息：文件名（Mac电脑上的Safari浏览器是一个值得注意的例外，它会展示文件名，但不会显示路径的其余部分）。



图19-1　热门浏览器和平台里的原生文件input示例：IE 8 Windows（上），Firefox 3 Windows（中）和Safari 4 Mac（下）

在很多情况下，一个自定义设计的文件输入控件在网站或应用程序里会比原生的控件更好用。

- 在一些网站里，文件输入控件的观感必须匹配用户界面设计的其余部分，或者它的尺寸必须加以控制才能纳入某个特定的布局里。
- 在一些案例里，“浏览”（Browse）或“选择文件”（Choose File）按钮带有图标，或者需要使用不同的用词来更准确地融入上下文（比如“选择图像”或者“上传个人资料图片”）

或者对不同的语言提供支持。

- 某些情况下，所选文件的反馈信息（包括文件名）必须可见，或者带有自定义的图标。
- 在一些案例里，用户能够上传多个文件，此时显示缩略图、文件大小或者展示上传过程实时反馈信息的进度条就很有帮助。

Flash等基于插件的解决方案通常用来创建自定义的文件输入控件，前面讨论过，插件的私有本质决定了它们不可能出现在所有平台和设备上。因此，我们倾向于用一些CSS和JavaScript来增强标准的HTML文件`input`，它的可访问性要广泛得多。

本章会分步介绍构建自定义文件输入控件的过程，此控件借助了原生的`input`，并用到一些创新手法。我们会在这一章的最后讨论一些更为复杂的情形，比如并列多个文件输入控件。

19.1　X 光透视

我们给某个社交网站做过一个让用户上传一张个人资料照片的表单。这个表单的文件输入控件是自定义的，以匹配网站的整体风格，如图19-2所示。

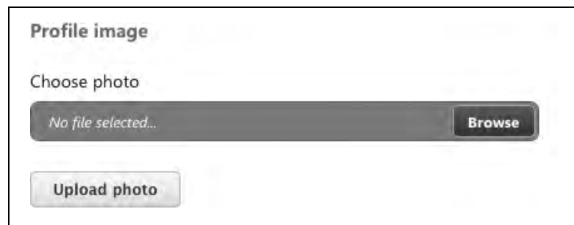


图19-2　自定义的文件输入控件

和原生的文件输入控件一样，用户点击“浏览”按钮时，会显示一个标准的操作系统对话框，如图19-3所示。

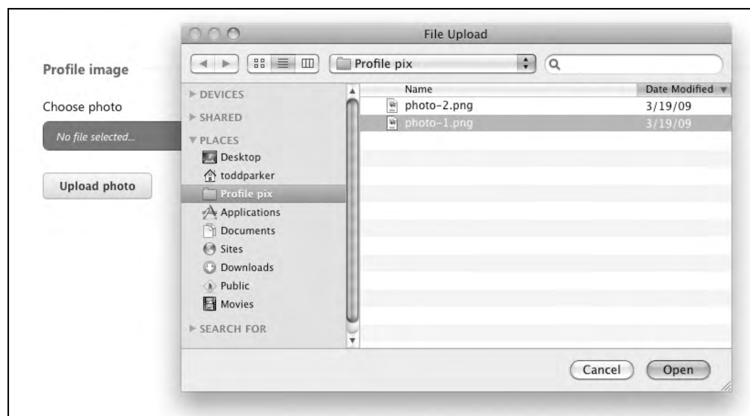


图19-3　点击“浏览”按钮后出现的原生浏览器对话框

在我们的目标设计里，自定义input会显示文件的名称和一个用于标识文件类型的图标。另外，把按钮文本从“浏览”换成“更改”（Change），虽然只是一个很小的变化，但却反映出已选择某个文件这一事实，如图19-4所示。



图19-4　某个选中文件的文件名和图标反馈信息

用X光“透视”这个设计后，为这个元素选择哪种基础标记就很明显了：HTML里上传文件的唯一标记就是文件input。

但是，原生的文件input无法通过添加样式匹配目标设计。为了解决这个问题，我们会隐藏这个原生文件input，使其不可见，然后在它的位置上生成外观与自定义组件相似的标记。只要点击这个自定义组件，就把click事件转移到原生的input上，由它打开操作系统的“浏览”对话框：本质上是“站在原生控件的肩膀上”打开“浏览”对话框。

一般来说，点击自定义组件时会用JavaScript触发原生文件input的click事件。但是，因为文件input直接与用户的文件系统交互，所以浏览器开发者内建了保护措施，以防止黑客利用JavaScript触发的click事件（有时也称为*劫持点击*“hijacking click”或者*点击劫持*“click-jacking”），所以需要一种巧妙的变通方法。

因为无法用程序触发原生文件input的click事件，所以我们会动态定位到input自身，确保它能获得*真正的*点击。为此，我们会使原生的文件input变得透明，并将它直接定位到用户光标和自定义文件input之间。用CSS把原生input的opacity设成0，使它不可见但仍可点击（使用display:none或visibility:hidden会导致它实际上不存在，因而无法点击）。这样用户就能直接点击原生的文件input，虽然看起来点的还是自定义文件输入控件，如图19-5所示。

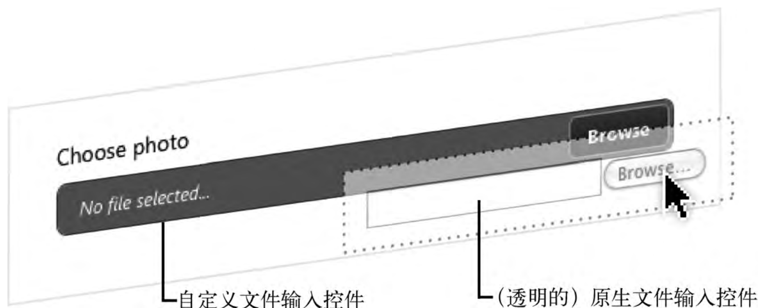


图19-5　原生文件输入控件被放在用户光标下方，通过点击可以打开浏览对话框

这样的做的优点是可以将原生文件input控件同时用于基本体验和增强体验，因为能借助它提供的所有功能和可访问性，而自定义文件输入控件其实只是一套视觉反馈机制而已。虽然原生input不可见，但它仍然可以通过键盘访问，因此无需给它添加任何可访问性功能或者ARIA属性。但是，我们会给自定义组件添加一个ARIA属性，让它对屏幕阅读器隐藏，因为它只对视力正常的用户有用。

使用原生文件input的基本体验是完美可用的，如图19-6所示。



图19-6　图像上传工具的基本体验

致　　谢

这种方法受到了Michael McGrady和Shaun Inman所做工作的启发，他们开发了一些复杂的CSS和JavaScript把原生文件input放到用户的光标下面，以创建自定义的文件输入控件。

www.shauninman.com/archive/2007/09/10/styling_file_inputs_with_css_and_the_dom

www.quirksmode.org/dom/inputfile.html

19.2　创建自定义的文件输入控件

有了总体方案之后，接下来我们看一看用于基本体验的基础标记，然后再添加标记、样式和脚本增强信息。

19.2.1　基础标记和样式

基础标记是一个type属性为file的标准input：

```
<input type="file" name="file" />
```

给这个input添加一个ID以关联它的label，同时作为一个挂钩（即一种容易找到它的方式）提供给我们的增强脚本：

```
<label for="file">Choose photo</label>
<input type="file" name="file" id="file" />
```

最后，用一个form标签围住这段标记，并添加一个类型为submit的标准input作为“上传照片”（Upload Photo）按钮，如图19-7所示。要正确提交文件输入控件的数据，form的开头标签必须包含method="post"和enctype="multi-part/form"这两个属性，这样表单才能够提交二进制文件：

```
<form action="profilePhoto.php" method="post" enctype="multipart/form-data">
```



图19-7　应用安全样式之前的基本体验

接下来，在基本样式表里添加安全样式，输出给所有用户。我们会加入一列简单的字体列表来设置网页的默认字体，并添加一条规则将label设置为display: block，使它位于文件input的上面，如图19-8所示。

```
body { font-family: "Segoe UI", Arial, sans-serif; }
label { display:block; }
```



图19-8　应用安全样式后的基本体验

19.2.2　增强标记和样式

建立基础标记之后，接下来编写自定义的增强标记和样式。

首先创建一个容器div来存放自定义标记，并赋予它customfile类和一个值为true的aria-hidden属性（稍后会给它添加样式，将它作为容纳反馈盒子和按钮的外部容器）：

```
<div class="customfile" aria-hidden="true"></div>
```

给这个容器添加两个span元素：第一个用于显示所选文件的图标和文字反馈信息，第二个会成为“浏览”按钮。默认情况下，原生文件input是空白的，因此初始的反馈消息是“没有选择文件……”（No file selected...），按钮的标签则是“浏览”：

```
<div class="customfile" aria-hidden="true">
  <span class="customfile-button">Browse</span>
  <span class="customfile-feedback">No file selected...</span>
</div>
```

给外部容器应用深灰的背景色、边框和内边距样式，并用CSS3的border-radius属性来圆化边角。同时设置cursor:pointer属性，以便用户看到手形图标，表示可以点击：

```
.customfile {
  width: 400px;
  background: #666;
  cursor: pointer;
  overflow: auto;
  padding: 2px;
  border: 1px solid #444;
  -moz-border-radius:7px;
  -webkit-border-radius:7px;
  border-radius:7px;
}
```

还要让“浏览”按钮浮动到反馈盒子的右侧，并给它添加渐变背景图像和圆化边角：

```
.customfile-button {
  border: 1px solid #999;
  background: #333 url(../images/bg-submit.gif) bottom repeat-x;
```

```
  color: #fff;
  font-weight: bold;
  float: right;
  width: 50px;
  padding: .3em .6em;
  text-align: center;
  text-decoration: none;
  font-size: 1.2em;
  -moz-border-radius:5px;
  -webkit-border-radius:5px;
  border-radius:5px;
}
```

空白状态（用户选择文件之前）的初始反馈文字为斜体，如图19-9所示。

```
.customfile-feedback {
  display: block;
  margin: 1px 1px 1px 5px;
  font-size: 1.2em;
  color: #fff;
  font-style: italic;
  padding: .3em .6em;
}
```



图19-9 没有选择文件时的增强体验

为了在用户把鼠标悬停于按钮上或用键盘使input获得焦点时提供视觉反馈，再创建一条通用样式规则来装饰悬停和焦点状态，然后添加一条点状轮廓线作为对通过键盘获得焦点的反馈，如图19-10所示。

```
.customfile-hover .customfile-button,
.customfile-focus .customfile-button {
  color:#111;
  background: #aaa url(../images/bg-btn.png) bottom repeat-x;
  border-color:#aaa;
  padding: .3em .6em;
}
.customfile-focus .customfile-button {
  outline: 1px dotted #ccc;
}
```



图19-10 通过键盘获得焦点的反馈示例：按钮的样式类似悬停状态，但多了一条点状的轮廓线

选择某个文件后，脚本会给反馈盒子再添加一个customfile-feedback-populated类，它覆盖了之前的一些样式，加粗反馈文字并添加了一个文件图标背景，如图19-11所示。

```
.customfile-feedback-populated {
  color: #fff;
  font-style: normal;
  font-weight: bold;
  padding-left: 20px;
  background: url(../images/icon-generic.gif) left 4px no-repeat;
}
```



图19-11 带有通用文件图标的文件反馈信息

默认情况下，我们会设置一个通用的文件背景图标，如果脚本识别出一个更加具体的文件类型，则样式表会用对应的图标覆盖这个通用图标。为了更新图标，脚本会解析所选文件的文件名来寻找它的扩展名，然后给反馈盒子动态添加一个此扩展名所对应的类。举个例子，一个名为photo.png的文件会被赋予`class="png"`。在增强样式表里，这些类所包含的样式规则会指向独一无二的背景图像。

如果想让一个图标代表多种文件类型，可以组合这些类选择器（PNG，JPG，等等），让它们共用一条样式规则。这里设置了4个常见的图像扩展名使用同一个图标。因为我们的界面关注的是选择一张照片，所以这些是我们首先要接受的扩展名，如图19-12所示。

```
.jpg, .gif, .png, .jpeg, .bmp {
  background-image: url(../images/icon-image.gif);
}
```



图19-12 图像的图标反馈

我们会把原有文件`input`的`opacity`设置为`0`，使它在大多数现代浏览器里不可见（后面会编写用于Internet Explorer 6的透明度脚本）。用户把鼠标悬停在自定义`input`上时，脚本会先把原生的文件`input`附加到`body`的末尾，然后再显示它，以确保将它正确放到相对于窗口的鼠标坐标上。把这个`input`的`z-index`设置成`99999`，确保它会被放在自定义`input`的上方（不信其他网页元素的`z-index`会大过它）：

```
.customfile-nativeinput {
  position: absolute;
  cursor: pointer;
  opacity: 0;
  z-index: 99999;
}
```

最后，给基础标记里原有的`label`添加样式来匹配我们的目标设计，因为在增强体验里它在页面上仍然可见：

```
label {
  font-size: 1.4em;
  display: block;
  margin: .5em 10px .5em 0;
}
```

19.2.3　自定义文件输入控件的脚本

调用增强脚本时要做的第一件事，就是给原生input添加customfilen-ativeinput类，以便给它应用样式。我们还会用jQuery的css方法让这个元素在Internet Explorer上不可见（Internet Explorer 6不支持CSS的opacity属性，因此jQuery会自动应用IE专有的filter属性达到同样的目的）：

```
var fileInput = $('#file')
  // 添加用于CSS的类
  .addClass('customfile-nativeinput')

  // 用jQuery的opacity方法在视觉上隐藏这个元素
  .css('opacity',0);
```

接下来，生成增强标记：容器div和两个span元素，并把它们添加到网页里的原生input之后。这个自定义文件输入控件默认是空白的（不带任何值），因此它会包含No file selected...（"没有选择文件……"）：

```
// 创建自定义控件的容器
var upload = $('<div class="customfile" aria-hidden="true"></div>');

// 创建自定义控件的按钮
var uploadButton = $('<span class="customfile-button">Browse</span>').
appendTo(upload);

// 创建自定义控件的反馈
var uploadFeedback = $('<span class="customfile-feedback">No file
selected...</span>').appendTo(upload);

// 把增强标记插入到原生文件输入控件之后
upload.insertAfter(fileInput);
```

为了把原生input放在用户光标的下方，脚本会绑定mousemove事件来探测鼠标何时位于自定义文件输入控件之上。然后会动态重定位原生input左上角屏幕坐标，把它放在用户光标和自定义input中间。用户点击自定义input的按钮时，他们点击的实际上是不可见的原生文件input里的按钮，进而打开浏览对话框：

```
// 在mousemove时保持原生的input在光标下方以捕捉点击
upload.mousemove(function(e){
  fileInput.css({

  // 定位到光标*Y*坐标之上3 px
  top: e.pageY - 3,

  // 定位到光标*X*坐标右侧20 px
```

```
    left: e.pageX - fileInput.outerWidth() + 20
  });
});
```

要确保这里的*x*和*y*重定位在所有浏览器和布局里都没问题，关键是要让原生的input始终相对于文档的body进行定位。脚本对此功能的实现方法，是在光标悬停到自定义控件上时，把原生input放到HTML文档的末尾。用户把鼠标移动到自定义控件上并触发mouseover事件时，把原生input的附加到body标签的末尾（用appendTo方法）。用户把鼠标移出自定义input会触发mouseout事件，再用insertBefore方法把原生文件input放回它在HTML源代码里的初始位置，即自定义代码之前：

```
upload
 // 在mouseover时把原生文件输入控件移到body末尾
 .mouseover(function(){
     fileInput.appendTo('body');
 })
 // 在mouseout时把原生文件输入控件移回原来的位置
 .mouseout(function(){
     fileInput.insertBefore(upload);
 });
```

现在，用户点击按钮时，原生的input会像往常一样打开操作系统的浏览对话框，脚本什么也不用管。

为了复制原生input的悬停和聚焦状态，要修改脚本来探测用户何时聚焦或悬停于原生的input上，然后用程序给自定义控件添加一个hover类，以便通过它给悬停状态添加样式。当用户把鼠标移出原生input并触发blur事件时，还要移除hover类，让自定义input的按钮恢复原来的样式。类似地，当用户用Tab键在网页上切换时，要检测原生input何时获得焦点，然后添加customfile-focus类来模拟自定义组件获得了焦点（从技术上看，焦点仍在原生input上）：

```
fileInput
  // 在mouseover时添加鼠标悬停类
  .mouseover(function(){
      upload.addClass('customfile-hover');
  })
  // 在mouseout时移除鼠标悬停类
  .mouseout(function(){
      upload.removeClass('customfile-hover');
  })
  // 在focus时添加键盘聚焦类
  .focus(function(){
      upload.addClass('customfile-focus');
  })
  // 在blur时移除键盘聚焦类
  .blur(function(){
      upload.removeClass('customfile-focus');
 })
```

还需要给自定义控件的反馈盒子添加文件名和图标。用户在浏览对话框里选择某个文件后，我们要从原生input触发的change事件中解析出它的新值。使用正则表达式能从完整的路径里提

取出文件名以更新反馈文字，然后根据扩展名生成恰当的图标class，添加customfile-feedback-populated类来切换反馈区域的样式，并让自定义文件输入控件的按钮显示出“更改”字样：

```
fileInput
  .change(function(){
      // 路径里最后一个反斜杠之后的文本就是文件名
      var fileName = $(this).val().split(/\\/).pop();
      // 路径里最后一个句号之后的文本就是扩展名
      var fileExt = 'customfile-ext-' +
      fileName.split('.').pop().toLowerCase();

      // 更新反馈信息
      uploadFeedback
          // 给容器添加类来展示反馈信息已生成的状态
          .addClass('customfile-feedback-populated');

          // 将反馈文字设置为文件名
          .text(fileName)

          // 添加文件扩展名类
          .addClass(fileExt)
    // 更改按钮的文本
    uploadButton.text('Change');
 });
```

最后，自定义文件输入控件应该具有原生input的所有功能，包括能用编程的方式禁用它。如果有必要阻止用户与自定义文件输入控件交互，那么可以给原生input添加disabled="disabled"属性，同时给自定义控件提供一种视觉样式，使其看上去处于禁用状态。（第17章详细介绍了如何处理自定义控件禁用问题。）

19.3　使用自定义文件输入控件脚本

本书附带的自定义文件输入控件演示和代码（可以在本书的网站上找到：www.filamentgroup.com/dwpe）引用了一个脚本：jQuery.fileinput.js。它能用本章描述的原则自动创建自定义文件输入控件。

要在你的网页里使用这个脚本，请下载并引用演示页里列出的那些文件，然后对网页里你想转变成自定义文件输入控件的原生文件input调用customFileInput()：

```
//把一个ID为"file"的input转换成自定义文件输入控件
$('input#file').customFileInput();
```

如果要定制网页里所有的文件input，可以修改选择器：

```
//转换type属性为"file"的所有input
$('input[type=file]').customFileInput();
```

在这个案例里，安全限制让我们别无选择，只能使用原生的文件input。这一章很好地说明了如何在增强体验过程中创造性地重复使用来自基础标记的原生元素，而不是用JavaScript再造其功能。这不仅简化了编程工作，还让我们能够利用原生元素的可访问性功能。

HTML5有更完善的文件上传机制

HTML5工作组已经编写了许多规范，在它们获得浏览器支持后就能提供更多用于文件输入和上传的原生特性与功能。这些规范包含了一个新的File API来提供许多标准的实现方法，否则这些功能只能通过Adobe Flash或Sun Java等私有插件才能实现。HTML5里一些值得注意的文件上传功能包括下面这些。

- 文件拖放选择：随着Web应用程序变得越来越无缝融入桌面环境，用户期望有朝一日能够把文件从操作系统窗口拖动到浏览器里实现上传。这项功能通常会在照片打印等网站上提供，因为用户必须上传照片才能下单打印。HTML5的dataTransfer和dragEvent接口提供了原生实现这一功能的标准。
- 跟踪上传进度：接受文件上传的Web应用程序有时会提供进度指示器来告知用户最新的上传状态。这一功能以前通常都交给Flash处理，因为大多数浏览器不提供有关上传进度的信息。HTML5规范已经草拟了一份跟踪上传进度的事件建议（包括onloadstart和loadprogress事件），它们让反馈信息在不使用私有插件的情况下成为可能。
- 用单个input上传多个文件： HTML5规范描述了原生文件输入元素的一个新属性：multiple。它允许用户在一个原生操作系统浏览对话框里选择多个文件。

这些功能很快就可以在浏览器里原生使用了。写作本书时，Firefox 3.5已经为大量HTML5的文件API提供了强有力的支持。

放眼未来

普遍可访问不仅是一个值得追求的目标，使用测试驱动的渐进增强方法时，它也是一个能达到的目标。

我们相信渐进增强会快速成为所有 Web 开发的新准则。互联网的功能越来越强大，基于 Web 的新设备不断涌现，随之而来的就是对在所有这些设备上使用新功能的期待。于是，对可访问性、移动设备支持和广泛浏览器兼容性的考量不再是一项“可做可不做的工作”。

渐进增强（更具体地说是测试驱动的、把可访问性纳入考虑范围的渐进增强）是唯一能应对千变万化的用户需求和设备的现实方法。另外，渐进增强还是一种面向未来的方法，它天生能够在所有采用 Web 标准的新版浏览器和设备上工作。

我们很高兴地看到，许多流行的 JavaScript 库正开始接受渐进增强和可访问性。随着普及率的提升，它们的开发者社区会创建出更多的预制插件，让开发工作变得更加轻松。举个例子，jQuery UI 团队正在把本书里描述的许多渐进增强技巧移植到即将发布的插件里，并且致力于在不久的将来完整支持 ARIA。

我们还期待未来有更多的 HTML5 和 CSS3 特性会得到广泛支持，使我们能用更少的 JavaScript 和更简洁的语义创建出更加强大的体验。让人高兴的是，这一天到来时，EnhanceJS 测试套件的可定制性会让它随需应变，让我们能够测试这些新功能并输出到能够处理它们的浏览器上，同时仍然照顾到所有的传统浏览器，向它们提供到处都能用的基本体验。

对于今天的项目，本书介绍的核心工具和技巧会让在设计和开发过程中满怀信心地采用渐进增强方法。X 光透视、最佳编程实践、能力测试脚本和本书附带的预制组件插件，能够确保你的项目服务于每一个人。

渐进增强的Web设计

"本书通俗易懂，示例网站和代码简直太棒了！渐进增强方法适合每个网站设计和开发人员使用。"

——Amazon读者评论

渐进增强是一种Web开发方式，致力于向最广大的潜在受众提供尽可能最优的体验，同时让编程和测试工作得到简化。无论用户是在iPhone上，还是在最新最潮的高端系统上浏览网站，甚至只是听着屏幕阅读器的朗读，他们获得的体验都应该是一致的，功能和特性也要尽可能完整。

本书由全球著名Web设计公司Filament集团两位创始人和两位开发主力联手打造，其中Scott Jehl还是jQuery团队成员。四位作者具有多年的网站设计和开发经验，曾为网站、无线设备、Web应用设计过众多高度实用的用户界面，受到了高度赞扬。本书展示了如何利用渐进增强方法开发网站，从而获得最佳用户体验。本书既是理解渐进增强原则和益处的实用指南，也是详细的案例分析，目的是向设计师以及开发人员传授何时、何地以及如何采用体现渐进增强的具体编程和脚本技巧。

本书的主要内容

- 常规的编程方法忽视了用户的实际需要，渐进增强提供更高的包容性和可访问性
- 分析复杂的界面设计，找出能普遍工作的潜在语义化HTML体验，然后安全地叠加上高级增强信息
- 独一无二的浏览器能力测试套件EnhanceJS，能帮你只将增强信息发送给有处理能力的设备
- 组合HTML、CSS和JavaScript来实现渐进增强的最佳实践，HTML5和CSS3的具体应用实例
- 支持WAI-ARIA和键盘操作
- 更多极具诱惑力的实战细节

New Riders

图灵社区：iTuring.cn

热线：(010)51095186转600

分类建议 计算机/网站开发

人民邮电出版社网址：www.ptpress.com.cn



ISBN 978-7-115-33839-6

定价：69.00元

欢迎加入

图灵社区

最前沿的IT类电子书发售平台

电子出版的时代已经来临。在许多出版界同行还在犹豫彷徨的时候，图灵社区已经采取实际行动拥抱这个出版业巨变。作为国内第一家发售电子图书的IT类出版商，图灵社区目前为读者提供两种DRM-free的阅读体验：在线阅读和PDF。

相比纸质书，电子书具有许多明显的优势。它不仅发布快，更新容易，而且尽可能采用了彩色图片（即使有的书纸质版是黑白印刷的）。读者还可以方便地进行搜索、剪贴、复制和打印。

图灵社区进一步把传统出版流程与电子书出版业务紧密结合，目前已实现作译者网上交稿、编辑网上审稿、按章发布的电子出版模式。这种新的出版模式，我们称之为"敏捷出版"，它可以让读者以较快的速度了解到国外最新技术图书的内容，弥补以往翻译版技术书"出版即过时"的缺憾。同时，敏捷出版使得作、译、编、读的交流更为方便，可以提前消灭书稿中的错误，最大程度地保证图书出版的质量。

最方便的开放出版平台

图灵社区向读者开放在线写作功能，协助你实现自出版和开源出版的梦想。利用"合集"功能，你就能联合二三好友共同创作一部技术参考书，以免费或收费的形式提供给读者。（收费形式须经过图灵社区立项评审。）这极大地降低了出版的门槛。只要你有写作的意愿，图灵社区就能帮助你实现这个梦想。成熟的书稿，有机会入选出版计划，同时出版纸质书。

图灵社区引进出版的外文图书，都将在立项后马上在社区公布。如果你有意翻译哪本图书，欢迎你来社区申请。只要你通过试译的考验，即可签约成为图灵的译者。当然，要想成功地完成一本书的翻译工作，是需要有坚强的毅力的。

最直接的读者交流平台

在图灵社区，你可以十分方便地写作文章、提交勘误、发表评论，以各种方式与作译者、编辑人员和其他读者进行交流互动。提交勘误还能够获赠社区银子。

你可以积极参与社区经常开展的访谈、审读、评选等多种活动，赢取积分和银子，积累个人声望。